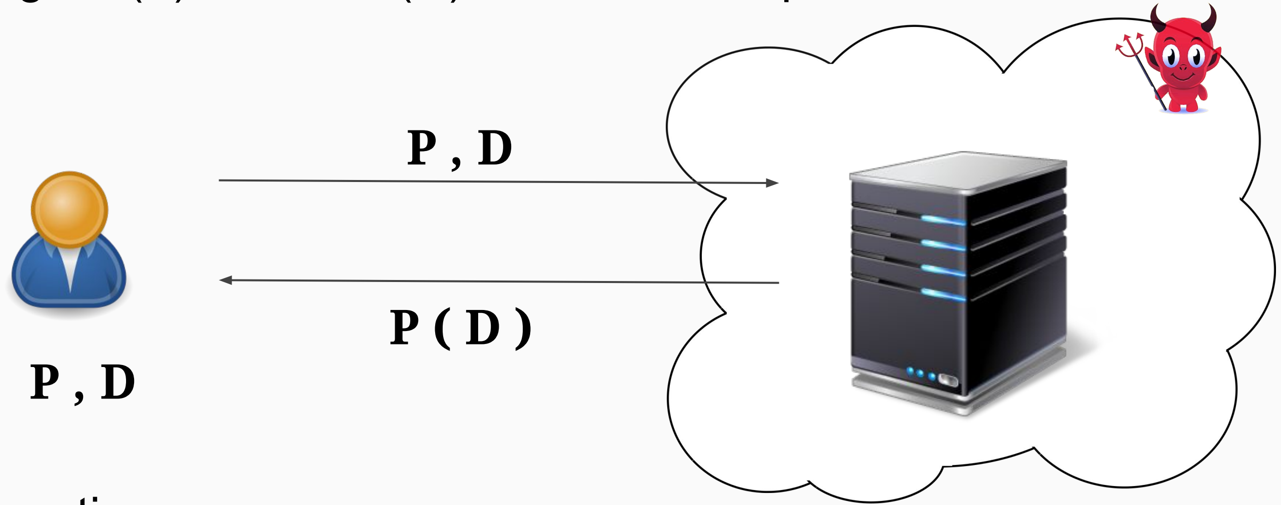
ZeroTrace: Oblivious Memory Primitives from Intel SGX

Sajin Sasy[1], Sergey Gorbunov[1] and Christopher Fletcher[2]

1 - University of Waterloo, 2 - University of Illinois at Urbana-Champaign

Secure Remote Computations

Alice with Program(P) and Data(D) wishes to compute P(D)

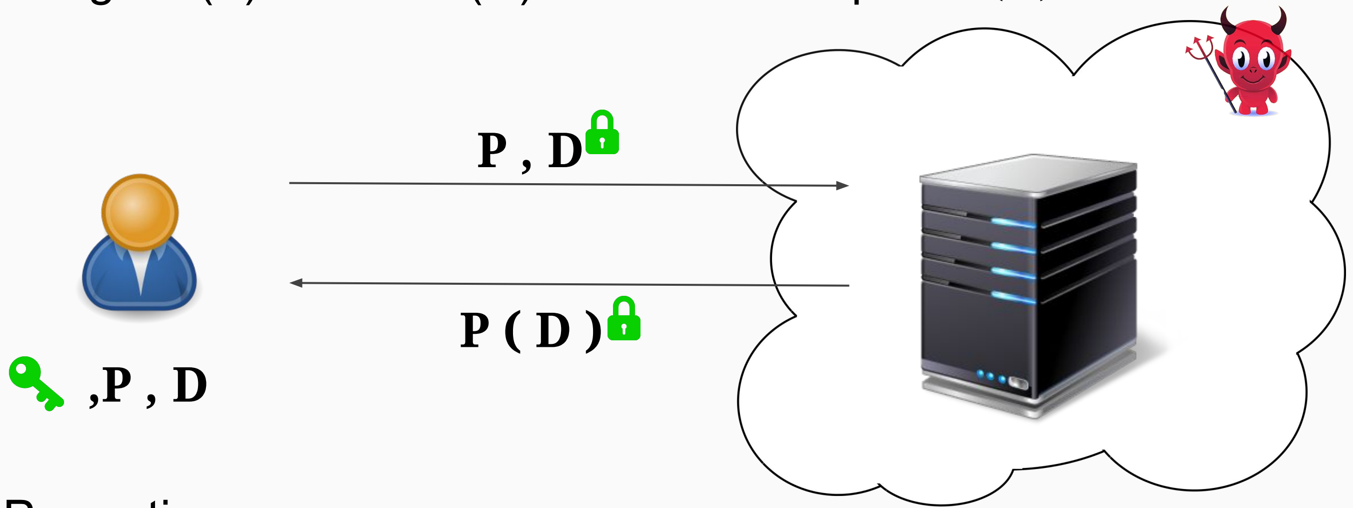


Desirable Properties :

- **Confidentiality** : The server learns nothing about D
- **Integrity** : The server can only return P(D) and no other function of D
- **Efficiency** : It executes in time close to natively executing P(D)

Solution in the ideal world

Alice with Program(P) and Data(D) wishes to compute P(D)



Desirable Properties :

- **Confidentiality** : The server learns nothing about D
- **Integrity** : The server can only return P(D) and no other function of D
- **Efficiency** : It executes in time close to natively executing P(D)

Real world tools and techniques

Software Solutions :

FHE [1] :

[1] - Gentry, Craig. *A fully homomorphic encryption scheme*.

Real world tools and techniques

Software Solutions :

FHE [1] :

Confidentiality	✓
Integrity	✗
Efficiency	✗

[1] - Gentry, Craig. *A fully homomorphic encryption scheme*.

Real world tools and techniques

Software Solutions :

FHE [1] :	
Confidentiality	✓
Integrity	✗
Efficiency	✗

ORAM [2] :	
Confidentiality	✓
Integrity	✓
Efficiency	✗

[1] - Gentry, Craig. *A fully homomorphic encryption scheme*.
[2] - Oded Goldreich and Rafail Ostrovsky. *Software protection and simulation on oblivious RAMs.*

Real world tools and techniques

Software Solutions :

FHE [1] :

Confidentiality	✓
Integrity	✗
Efficiency	✗

ORAM [2] :

Confidentiality	✓
Integrity	✓
Efficiency	✗

Hardware Solutions :

Intel TPM+TXT [3] :

Confidentiality	✗
Integrity	✓
Efficiency	✓

[1] - Gentry, Craig. *A fully homomorphic encryption scheme*.
[2] - Oded Goldreich and Rafail Ostrovsky. *Software protection and simulation on oblivious RAMs.*
[3] - Scarlata, Vincent, et al. *TPM virtualization: Building a general framework.*

Real world tools and techniques

Software Solutions :

FHE [1] :

Confidentiality	✓
Integrity	✗
Efficiency	✗

ORAM [2] :

Confidentiality	✓
Integrity	✓
Efficiency	✗

Hardware Solutions :

Intel TPM+TXT [3] :

Confidentiality	✗
Integrity	✓
Efficiency	✓

Intel SGX [4] :

Confidentiality	✓
Integrity	✓
Efficiency	✓

[1] - Gentry, Craig. *A fully homomorphic encryption scheme*.
[2] - Oded Goldreich and Rafail Ostrovsky. *Software protection and simulation on oblivious RAMs.*
[3] - Scarlata, Vincent, et al. *TPM virtualization: Building a general framework.*
[4] - Anati, Ittai, et al. *Innovative technology for CPU based attestation and sealing*.

Real world tools and techniques

Software Solutions :

FHE [1] :

Confidentiality	✓
Integrity	✗
Efficiency	✗

ORAM [2] :

Confidentiality	✓
Integrity	✓
Efficiency	✗

Hardware Solutions :

Intel TPM+TXT [3] :

Confidentiality	✗
Integrity	✓
Efficiency	✓

Intel SGX [4] :

Confidentiality	✗ [5,6,7,8,9,10]
Integrity	✓
Efficiency	✓

[5] - Xu, Yuanzhong et al. *Controlled-channel attacks: Deterministic side channels for untrusted operating systems.*
[6] - Shinde, Shweta, et al. *Preventing page faults from telling your secrets.*
[7] - Lee, Sangho, et al. *Inferring fine-grained control flow inside SGX enclaves with branch shadowing.*
[8] - Brasser, Ferdinand, et al. *Software Grand Exposure: SGX Cache Attacks Are Practical.*
[9] - Moghimi, Ahmad et al. *Cachezoom: How SGX amplifies the power of cache attacks.*
[10] - Van Bulck, Jo, et al. *Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution.*

Our Goal

Can we design a solution that meets all the three desirable properties of secure remote computation ?

Yes, ZeroTrace.

Our Approach :

Privacy of ORAM

+

Efficiency of SGX

Outline

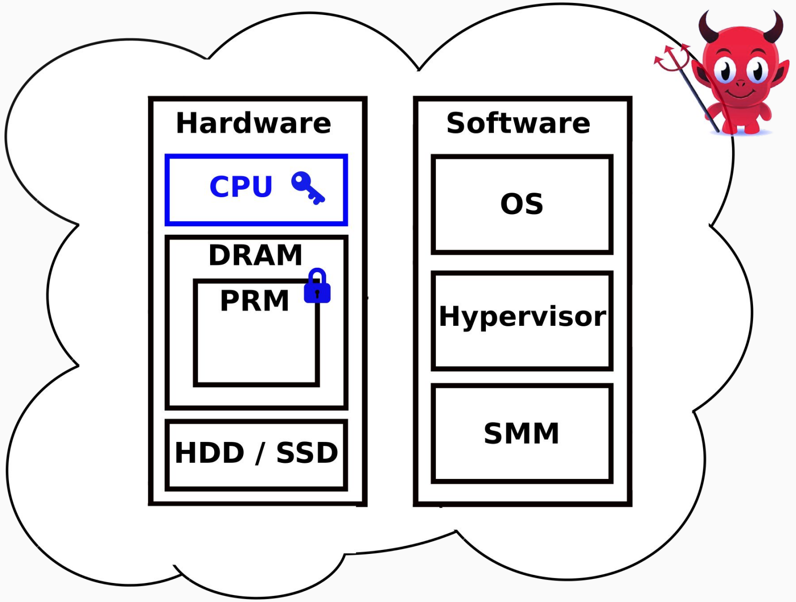
1. Secure Remote Computation ✓
2. Preliminaries :
   - Intel SGX
   - ORAM
3. ZeroTrace Architecture
4. Evaluation

Preliminaries

Intel SGX Background

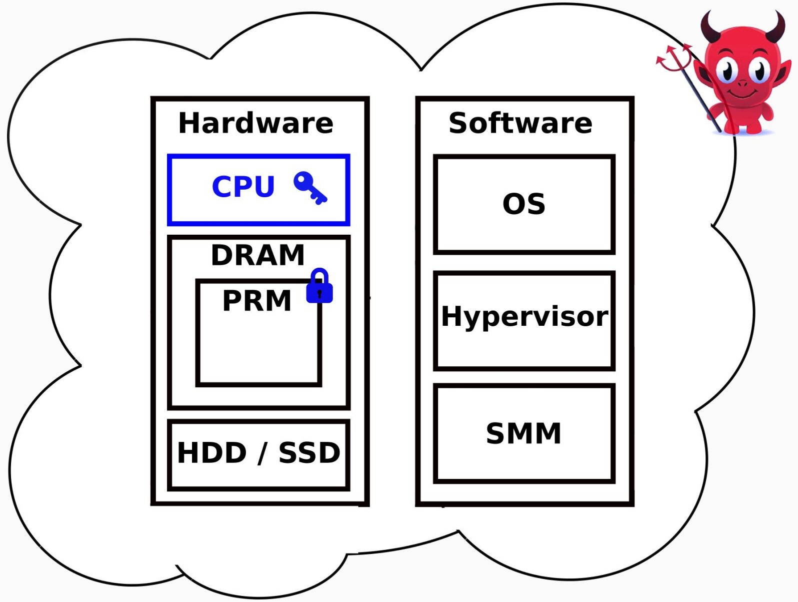
Intel SGX - Software Guard eXtensions

- x86 instructions extension



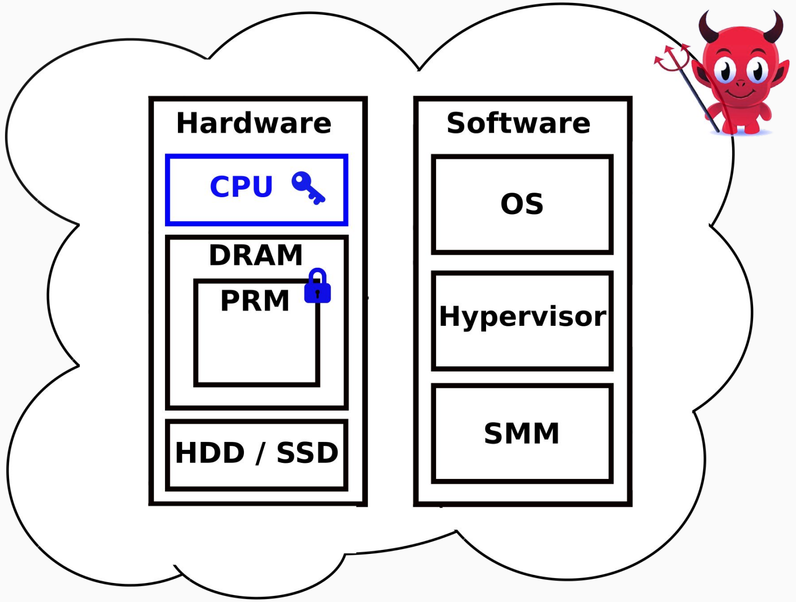
Intel SGX - Software Guard eXtensions

- x86 instructions extension
- Trusted processor fused with secret keys



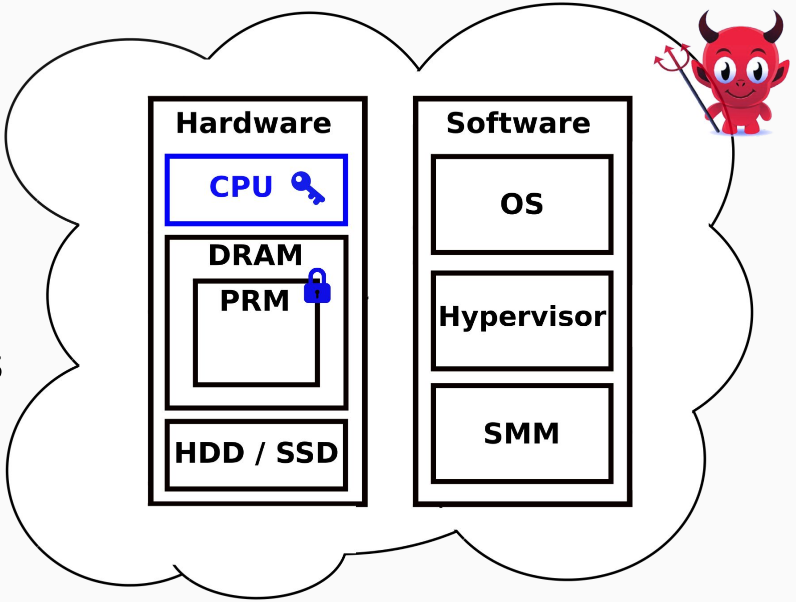
Intel SGX - Software Guard eXtensions

- x86 instructions extension
- Trusted processor fused with secret keys
- Processor Reserved Memory (**PRM**) set aside securely at boot



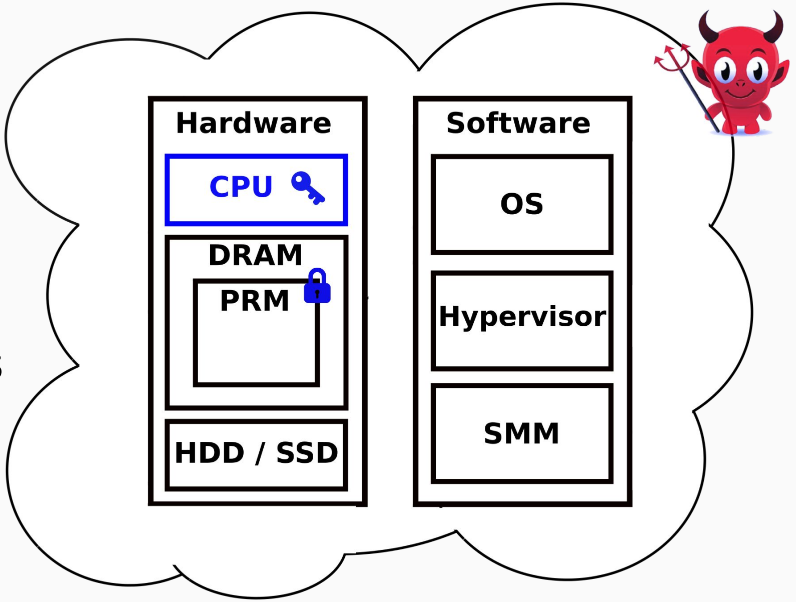
Intel SGX - Software Guard eXtensions

- x86 instructions extension
- Trusted processor fused with secret keys
- Processor Reserved Memory (**PRM**) set aside securely at boot
- Secure virtual containers called **enclaves**

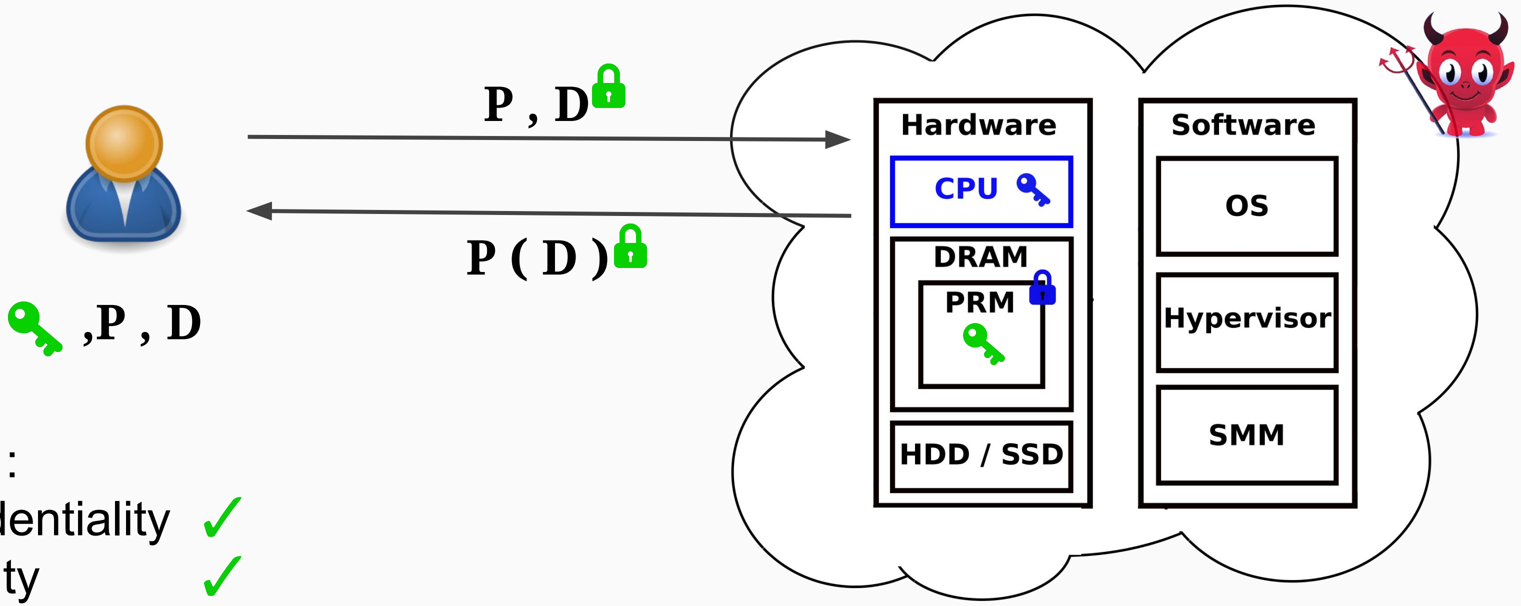


Intel SGX - Software Guard eXtensions

- x86 instructions extension
- Trusted processor fused with secret keys
- Processor Reserved Memory (**PRM**) set aside securely at boot
- Secure virtual containers called **enclaves**
- “Secure as long as processor isn’t physically broken into.”



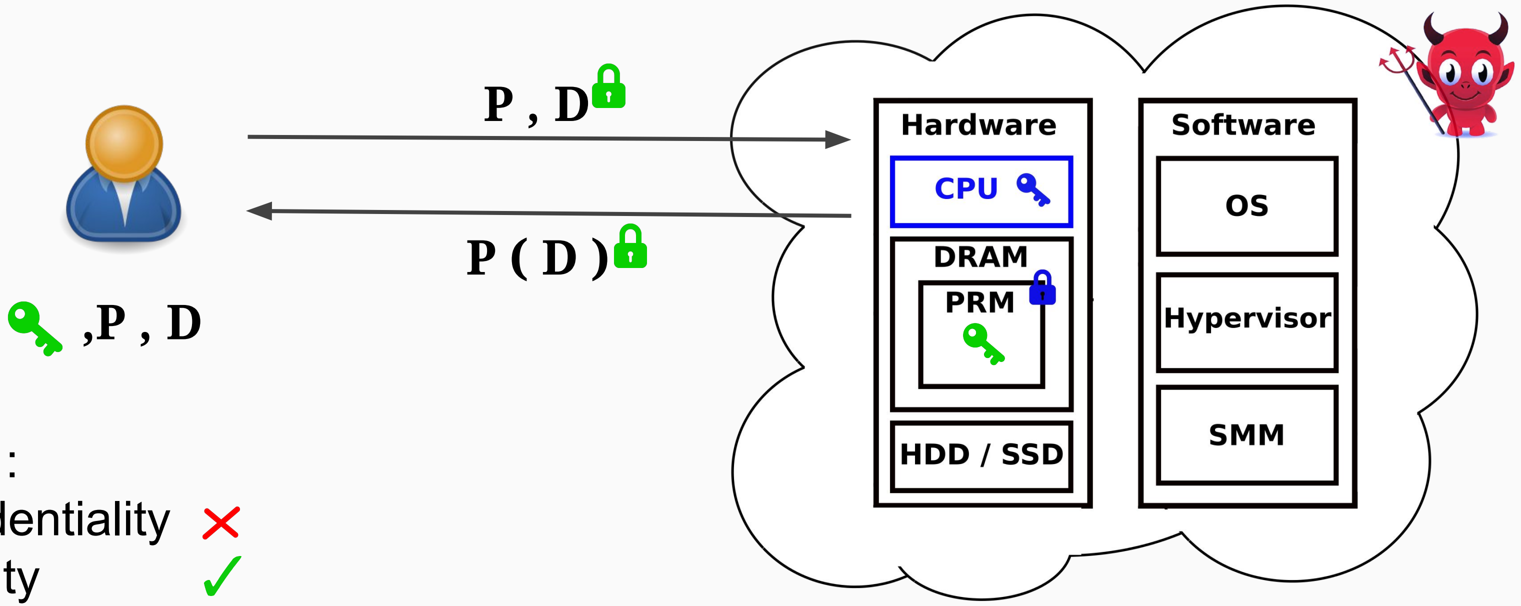
Secure Remote Computations with Intel SGX



Properties :

- Confidentiality ✓
- Integrity ✓
- Efficiency ✓

Secure Remote Computations with Intel SGX



Properties :

- Confidentiality ✗
- Integrity ✓
- Efficiency ✓

Security Limitations of SGX

Research has shown that SGX is susceptible to side channel attacks.
These attacks enable an adversary to extract secret data from enclaves !

- Page Fault Attacks [1,2]
- Branch Shadowing Attack [3]
- Cache Attacks [4,5]
- Data Access Pattern Attacks [1,6]

[1] - Xu, Yuanzhong, Weidong Cui, and Marcus Peinado. *Controlled-channel attacks: Deterministic side channels for untrusted operating systems.* 2015.
[2] - Shinde, Shweta, et al. *Preventing page faults from telling your secrets*. 2016.
[3] - Lee, Sangho, et al. *Inferring fine-grained control flow inside SGX enclaves with branch shadowing*. 2016.
[4] - Brasser, Ferdinand, et al. *Software Grand Exposure: SGX Cache Attacks Are Practical.* 2017.
[5] - Moghimi, Ahmad, Gorka Irazoqui, and Thomas Eisenbarth. *Cachezoom: How SGX amplifies the power of cache attacks*. 2017.
[6] - Van Bulck, Jo, et al. *Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution.* 2017.

Functional Limitations of SGX

- Effective PRM limited to 90 MB (with expensive cost for paging)
- No direct IO / syscalls
- Expensive context switching due to Asynchronous Enclave Exits (AEX)

Outline

1. Secure Remote Computation ✓
2. Preliminaries :
   - Intel SGX - Lightning Tour ✓
   - ORAM
3. ZeroTrace Architecture
4. Evaluation

Preliminaries

ORAM or Oblivious RAM

Oblivious RAM [1]

- Cryptographic primitive designed to hide memory access patterns
- All ORAMs fundamentally require a probabilistic encryption schema

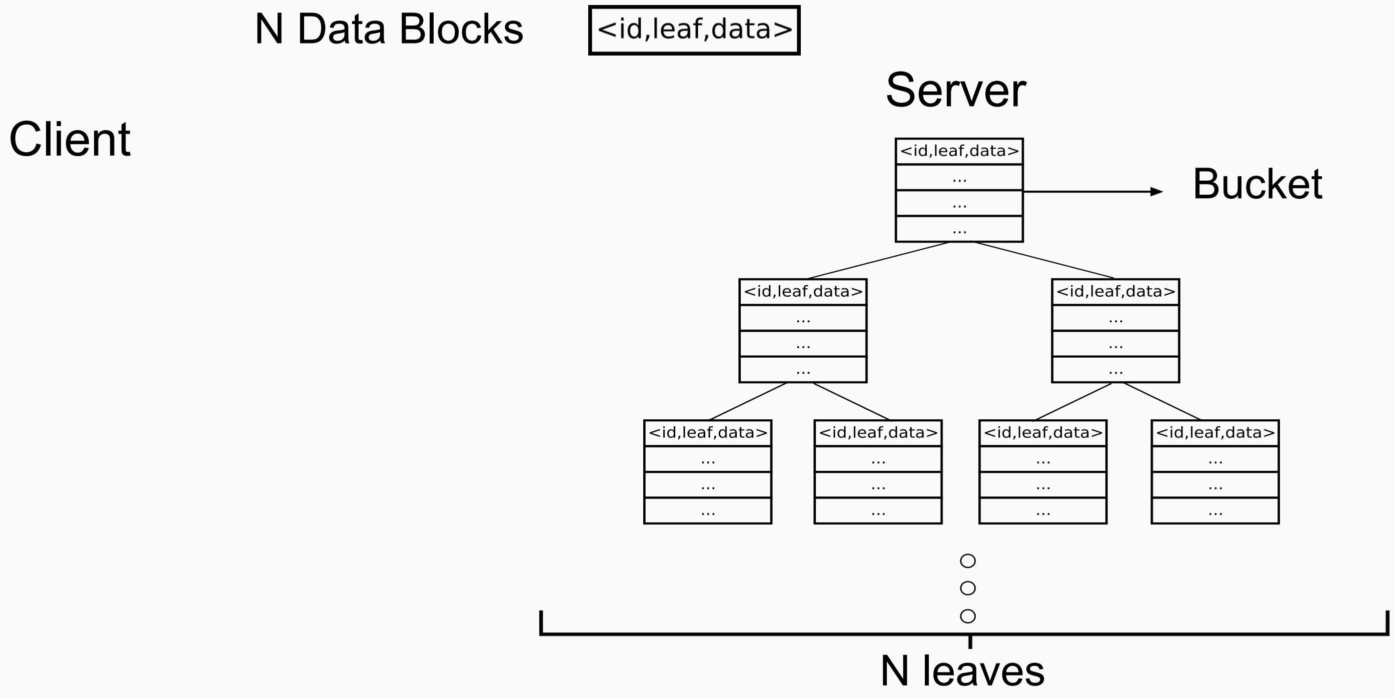
[1] - Goldreich, Oded, and Rafail Ostrovsky. *Software protection and simulation on oblivious RAMs.* 1996

Tree based ORAMs [1,2,3]

Data Blocks $\boxed{\text{<id,leaf,data>}}$

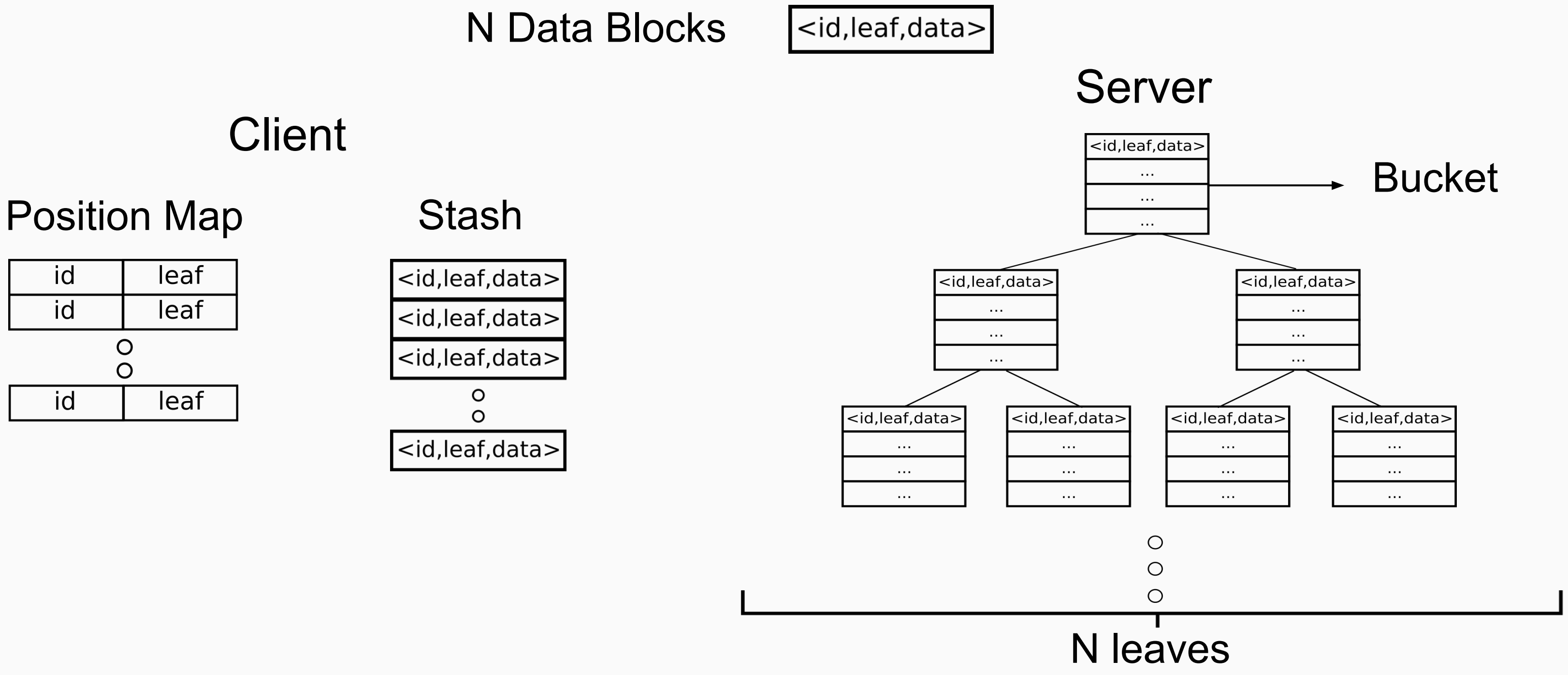
[1] - Shi, Elaine, et al. *Oblivious RAM with O ((logN) 3) Worst-Case Cost.* 2011.
[2] - Stefanov, Emil, et al. *Path ORAM: an extremely simple oblivious RAM protocol.* 2013.
[3] - Wang, Xiao, Hubert Chan, and Elaine Shi. *Circuit ORAM: On tightness of the Goldreich-Ostrovsky lower bound.* 2015.

Tree based ORAMs [1,2,3]



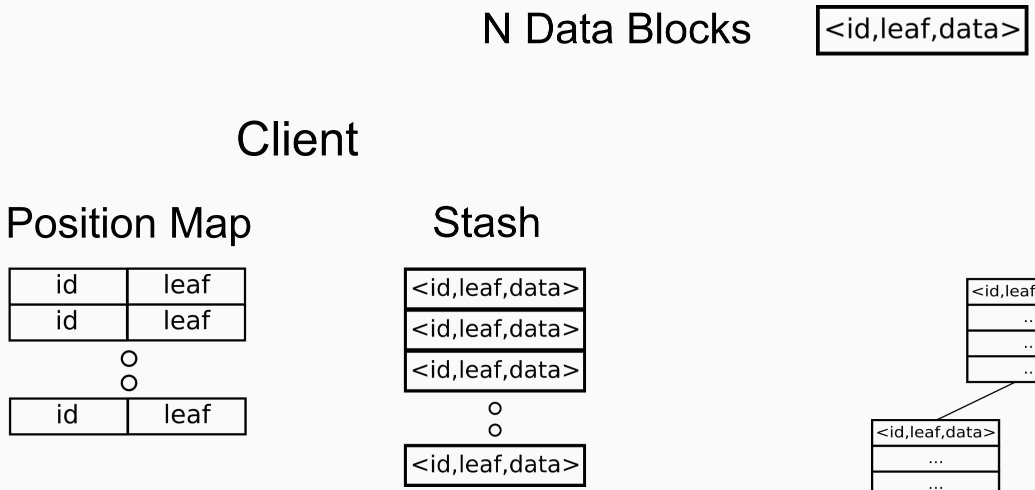
[1] - Shi, Elaine, et al. *Oblivious RAM with O ((logN) 3) Worst-Case Cost.* 2011.
[2] - Stefanov, Emil, et al. *Path ORAM: an extremely simple oblivious RAM protocol.* 2013.
[3] - Wang, Xiao, Hubert Chan, and Elaine Shi. *Circuit ORAM: On tightness of the Goldreich-Ostrovsky lower bound.* 2015.

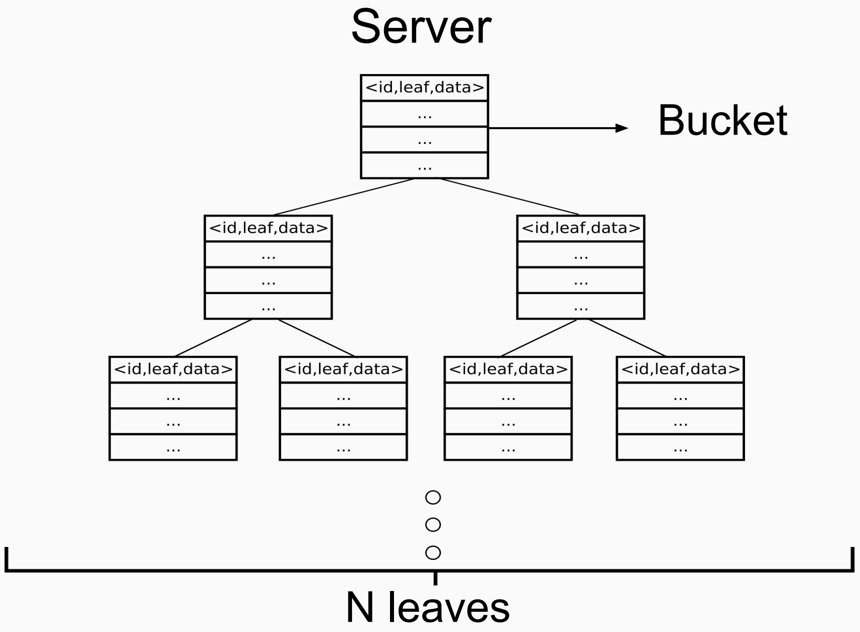
Tree based ORAMs [1,2,3]



[1] - Shi, Elaine, et al. *Oblivious RAM with O ((logN) 3) Worst-Case Cost.* 2011.
[2] - Stefanov, Emil, et al. *Path ORAM: an extremely simple oblivious RAM protocol.* 2013.
[3] - Wang, Xiao, Hubert Chan, and Elaine Shi. *Circuit ORAM: On tightness of the Goldreich-Ostrovsky lower bound.* 2015.

Tree based ORAMs [1,2,3]





Invariant : A block in the system will always reside in either the local stash, or in its path to the leaf label on the server tree.

[1] - Shi, Elaine, et al. *Oblivious RAM with O ((logN) 3) Worst-Case Cost.* 2011.
[2] - Stefanov, Emil, et al. *Path ORAM: an extremely simple oblivious RAM protocol.* 2013.
[3] - Wang, Xiao, Hubert Chan, and Elaine Shi. *Circuit ORAM: On tightness of the Goldreich-Ostrovsky lower bound.* 2015.

Tree based ORAMs - Access

① **Fetch leaf for id = 7**

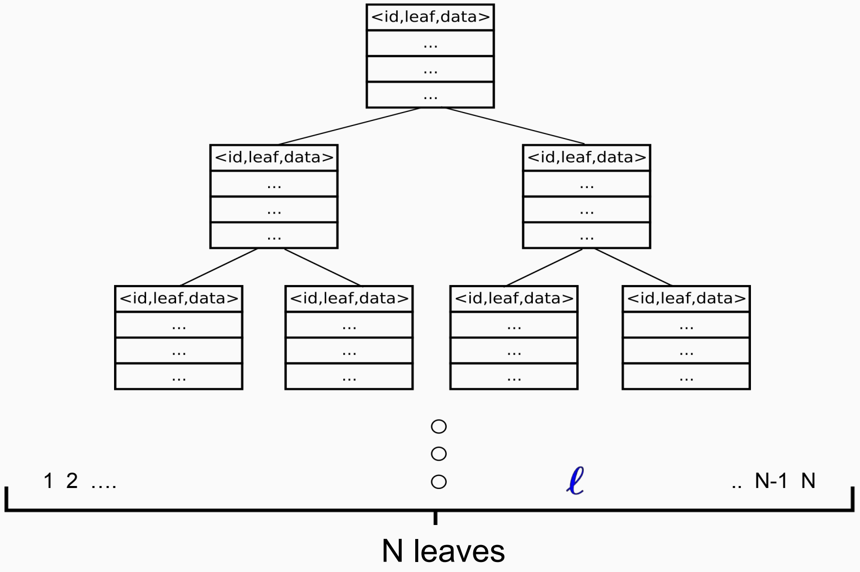
Client

Request ID : 7

Position Map

id	leaf
7	ℓ
○	○
id	leaf

Server



Tree based ORAMs - Access

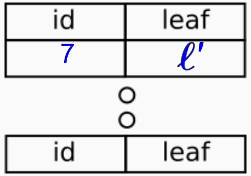
① Fetch leaf for id = 7

② **Sample new leaf ℓ'**

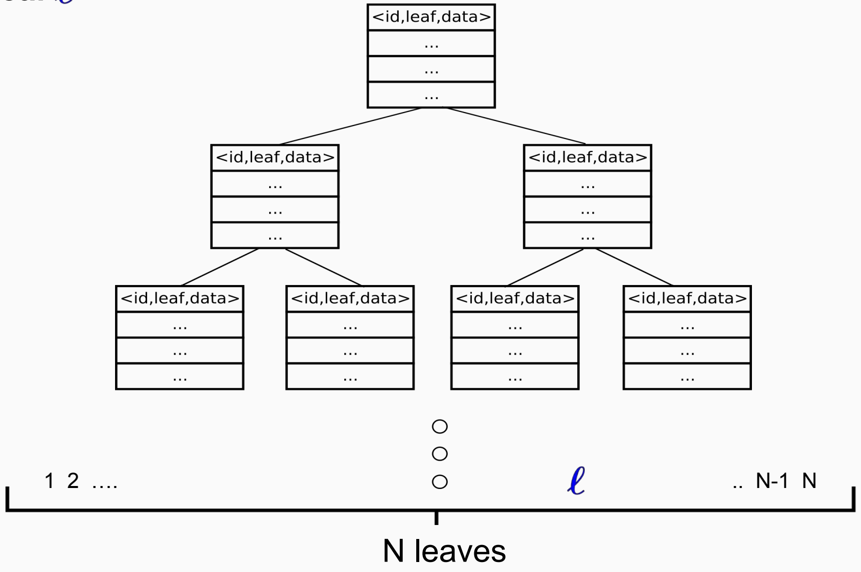
Client

Request ID : 7

Position Map

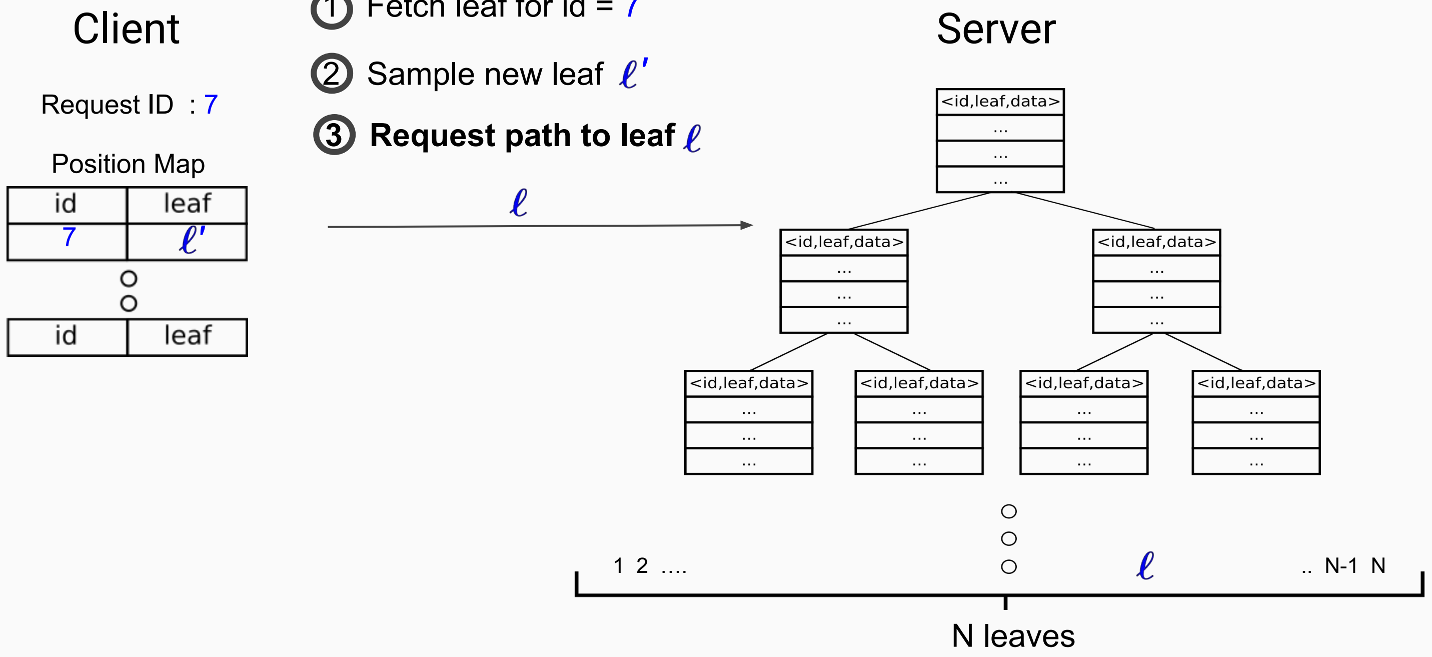


Server



Tree based ORAMs - Access

1. Fetch leaf for id = 7
2. Sample new leaf ℓ'
3. **Request path to leaf ℓ**

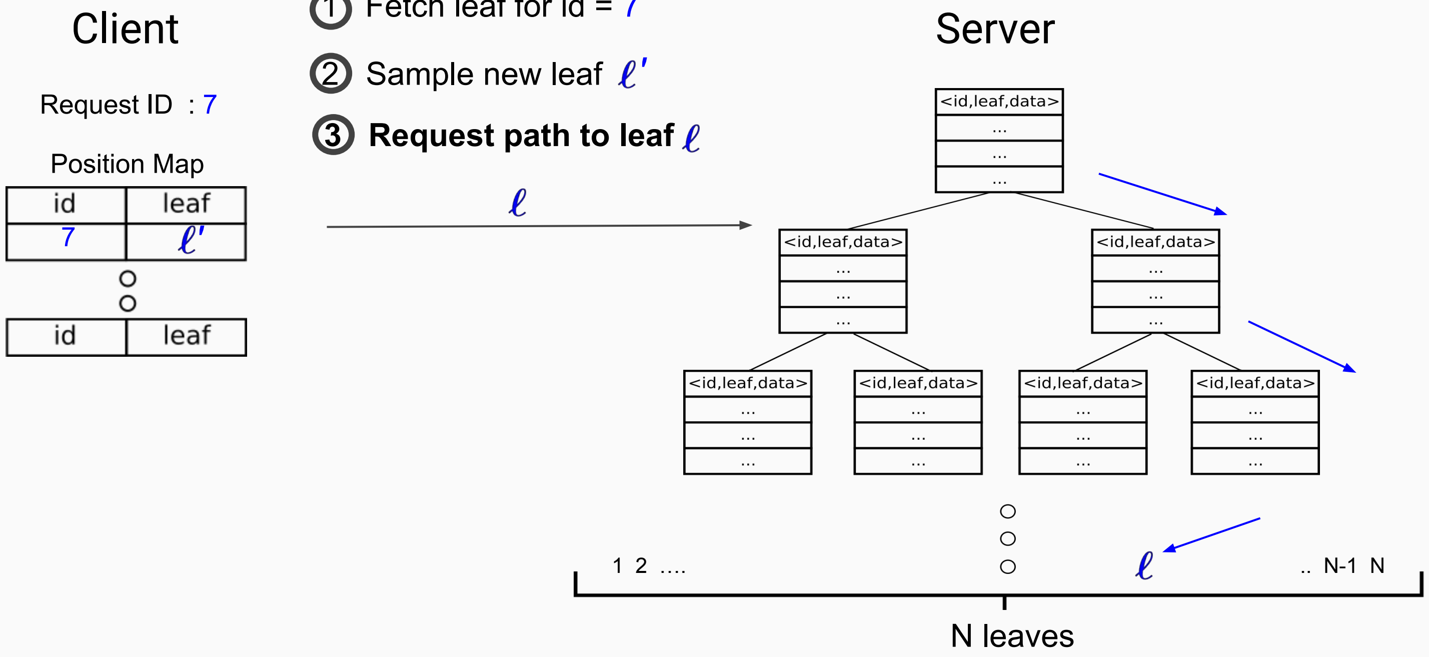


Tree based ORAMs - Access

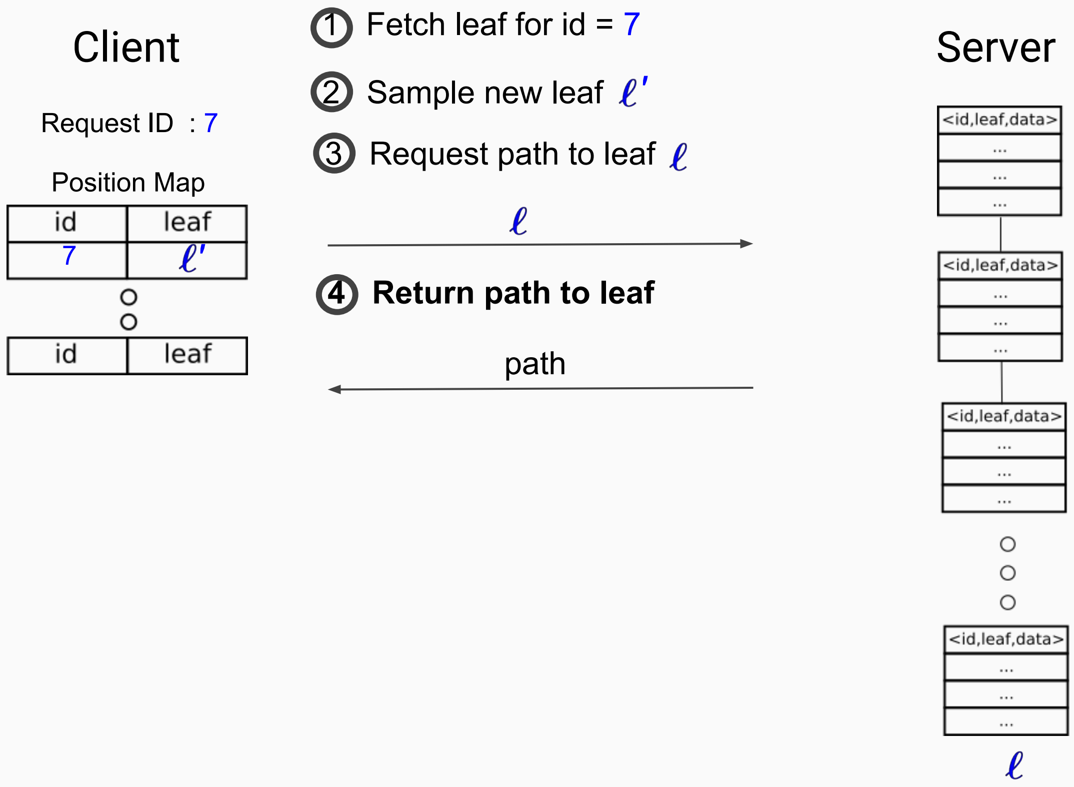
① Fetch leaf for id = 7

② Sample new leaf ℓ'

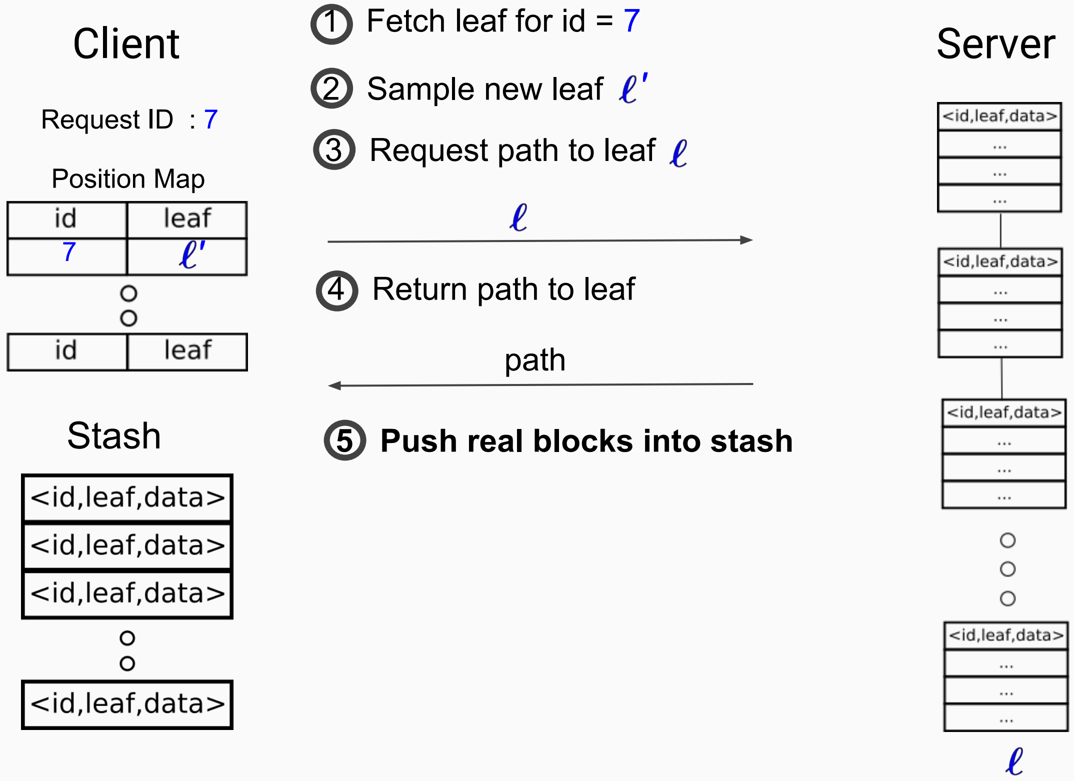
③ **Request path to leaf ℓ**



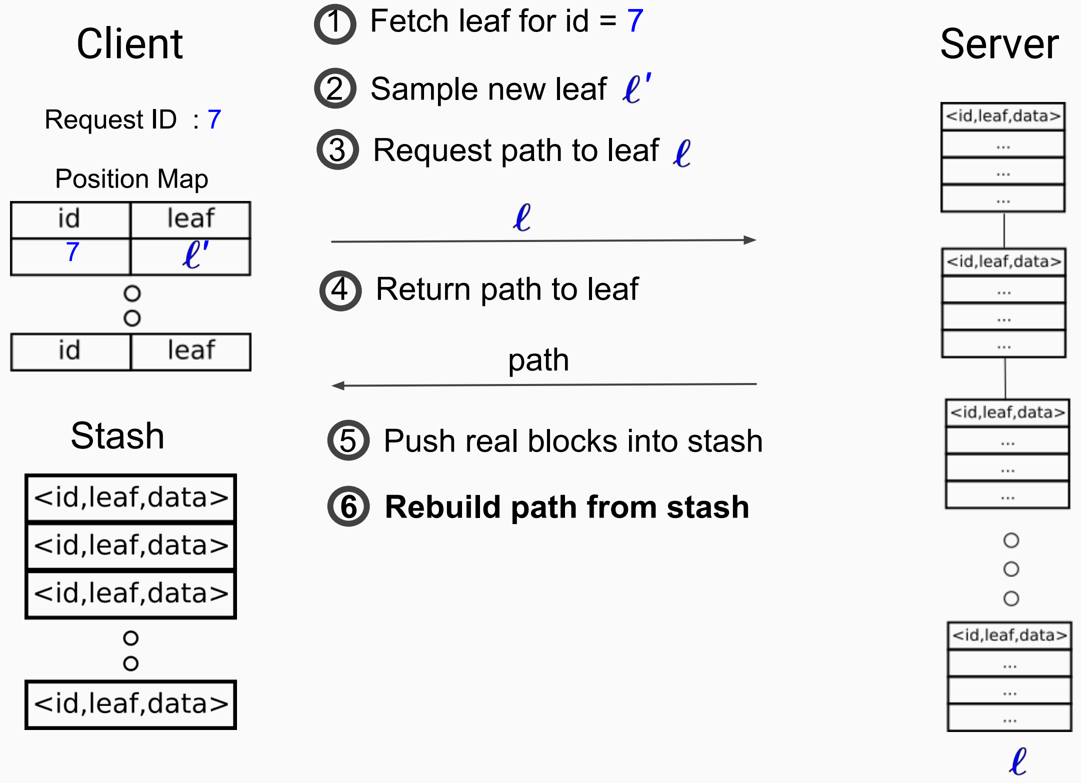
Tree based ORAMs - Access



Tree based ORAMs - Access



Tree based ORAMs - Access



① Fetch leaf for id = 7

② Sample new leaf ℓ'

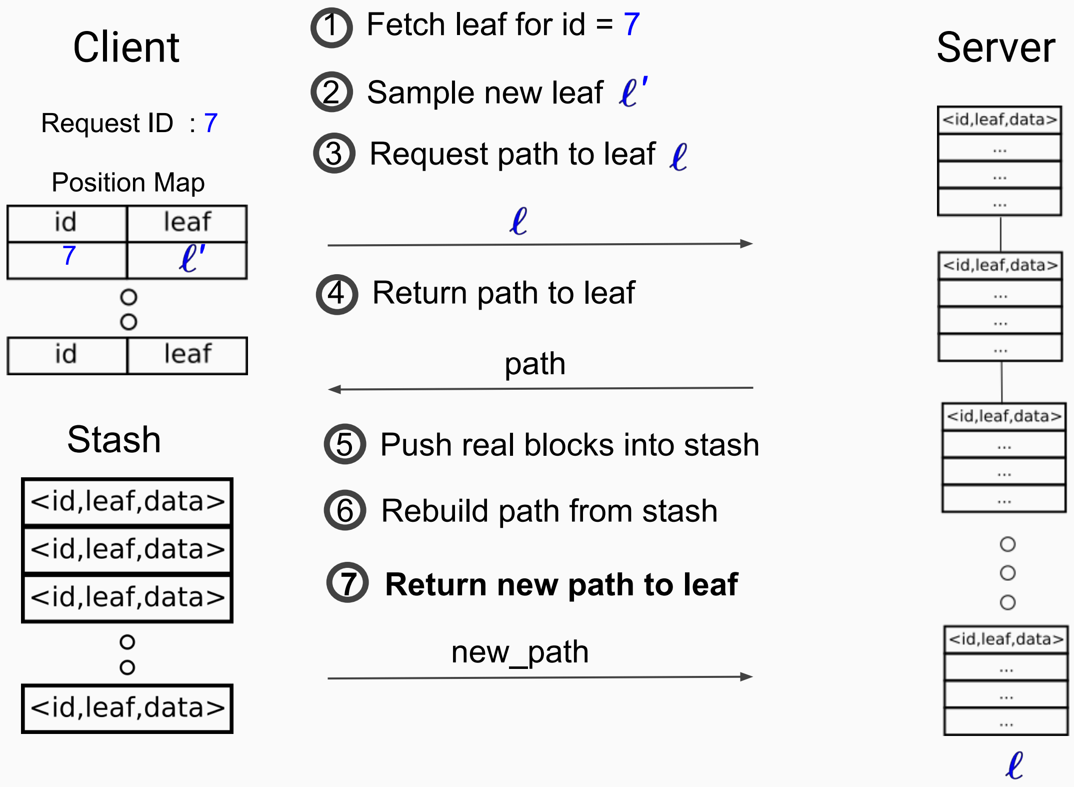
③ Request path to leaf ℓ

④ Return path to leaf

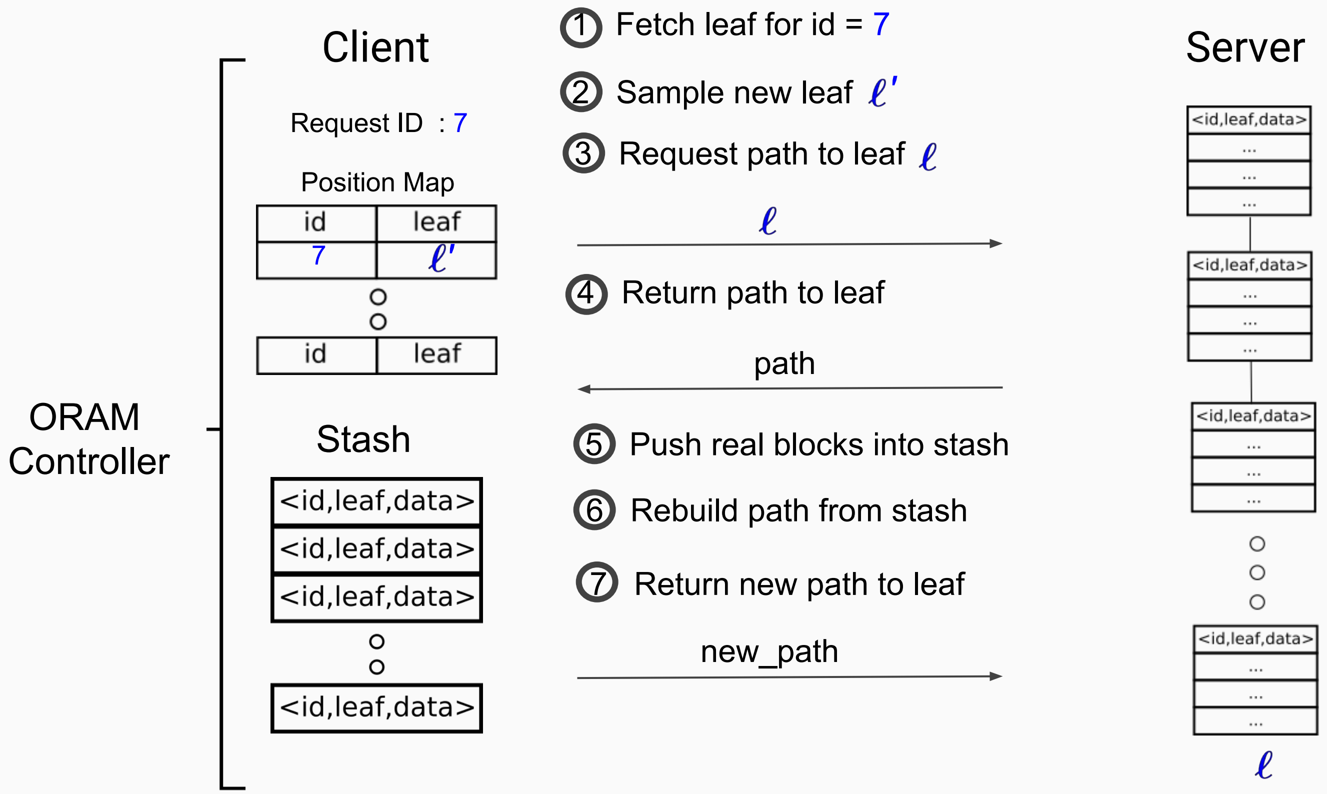
⑤ Push real blocks into stash

⑥ **Rebuild path from stash**

Tree based ORAMs - Access



Tree based ORAMs - Access

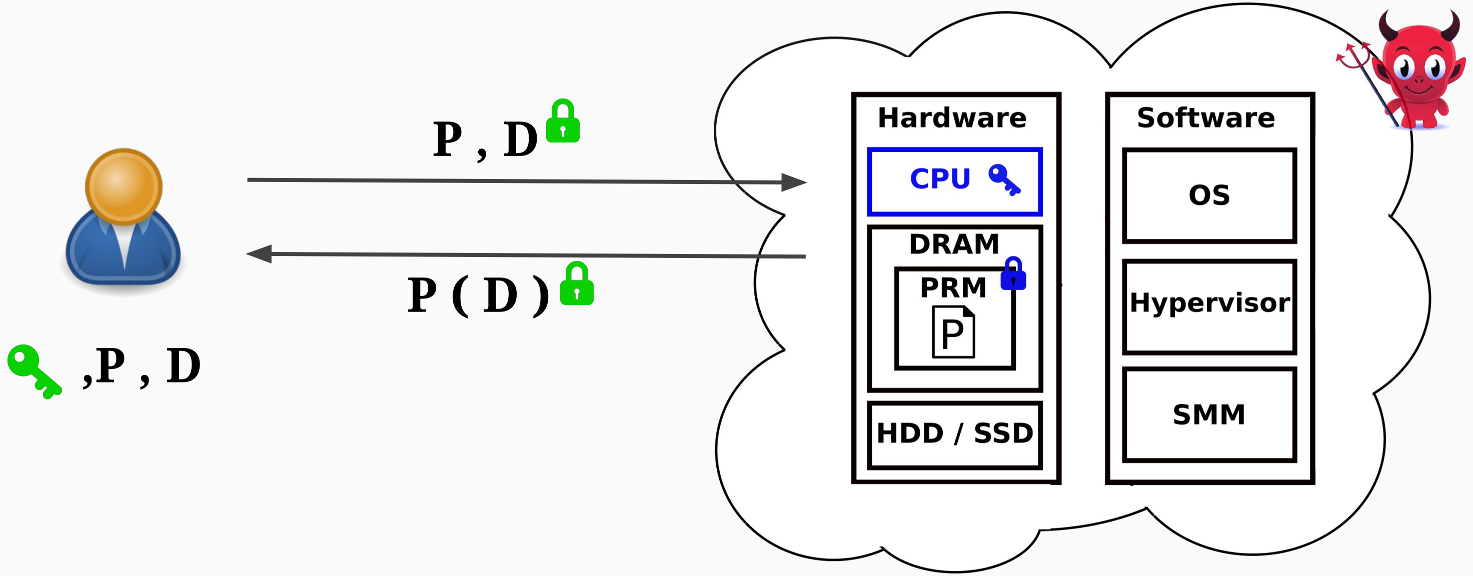


Outline

1. Secure Remote Computation ✓
2. Preliminaries :
    - Intel SGX - Lightning Tour ✓
    - ORAM ✓
3. ZeroTrace Architecture
4. Evaluation

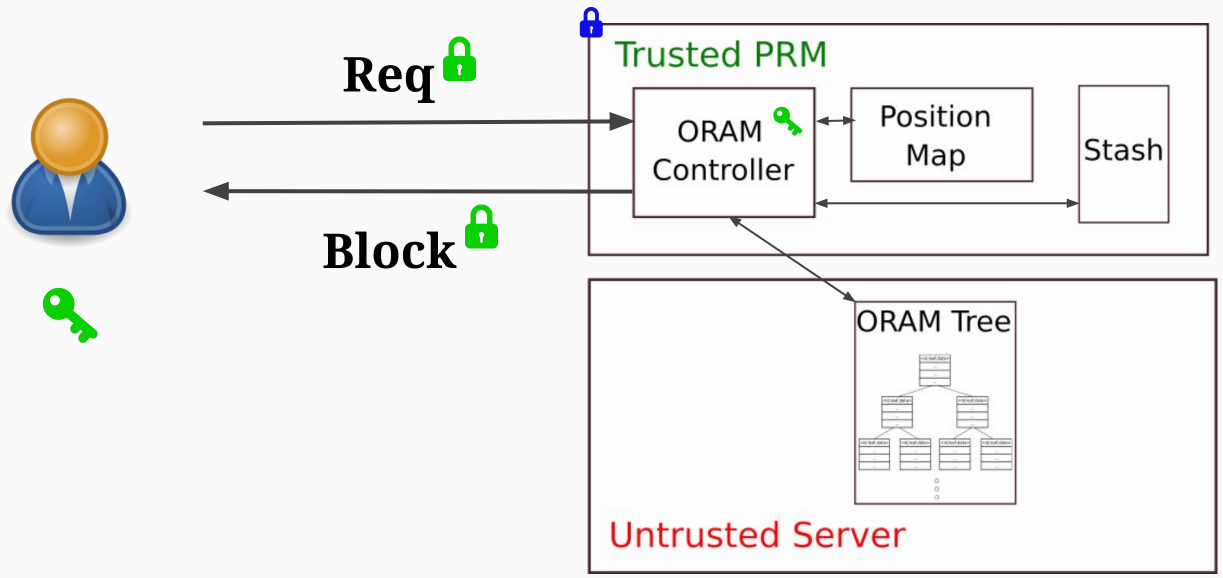
ZeroTrace Architecture

ZeroTrace Threat Model

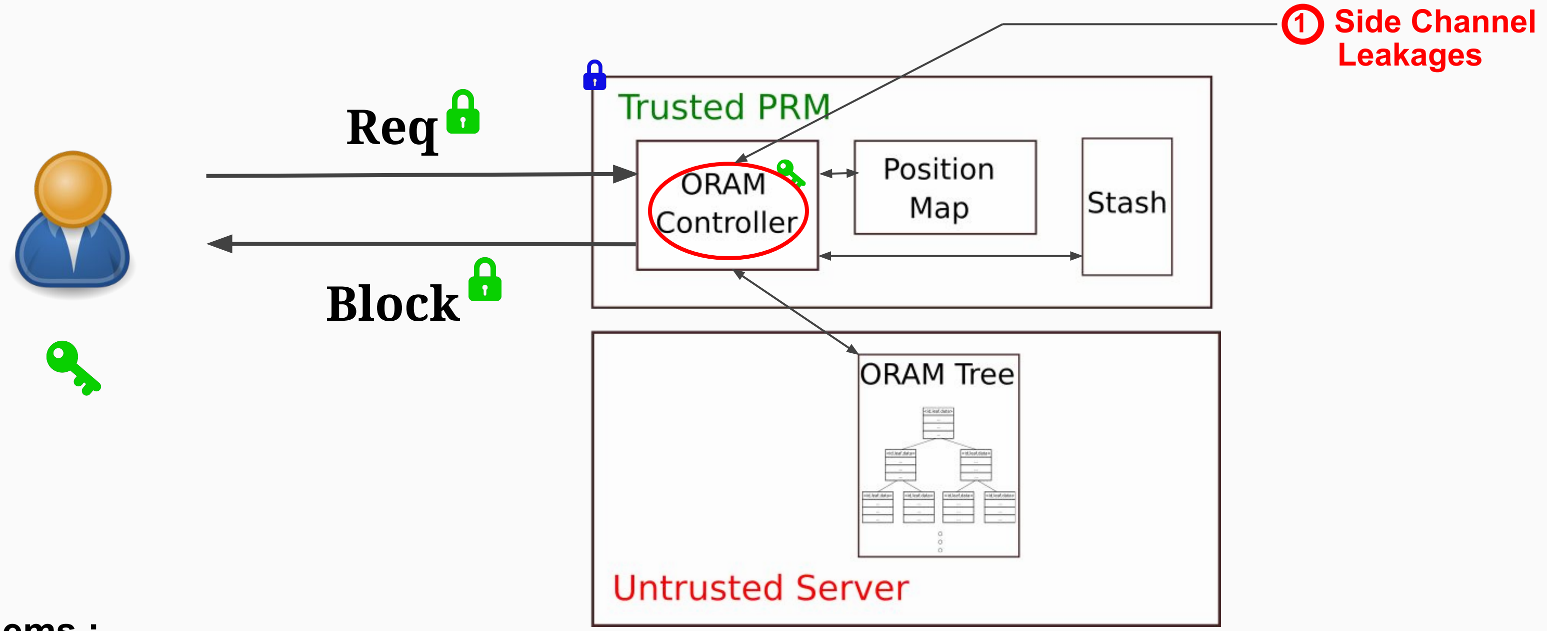


- By default we consider a malicious active adversary
- Everything on the server stack except the processor is untrusted

Attempt 1


Req
Block
Trusted PRM
ORAM Controller
Position Map
Stash
ORAM Tree
Untrusted Server

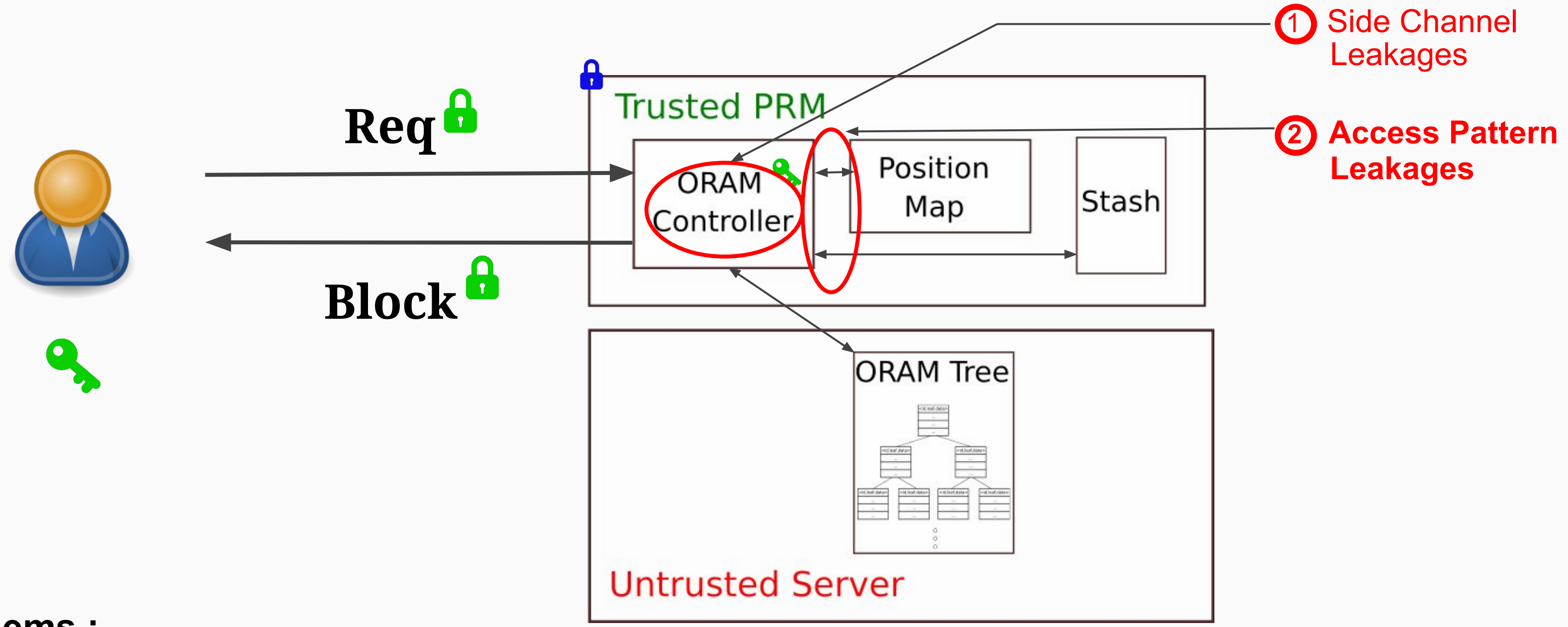
Attempt 1



Problems :

1. **Controller code susceptible to side channel leakages**

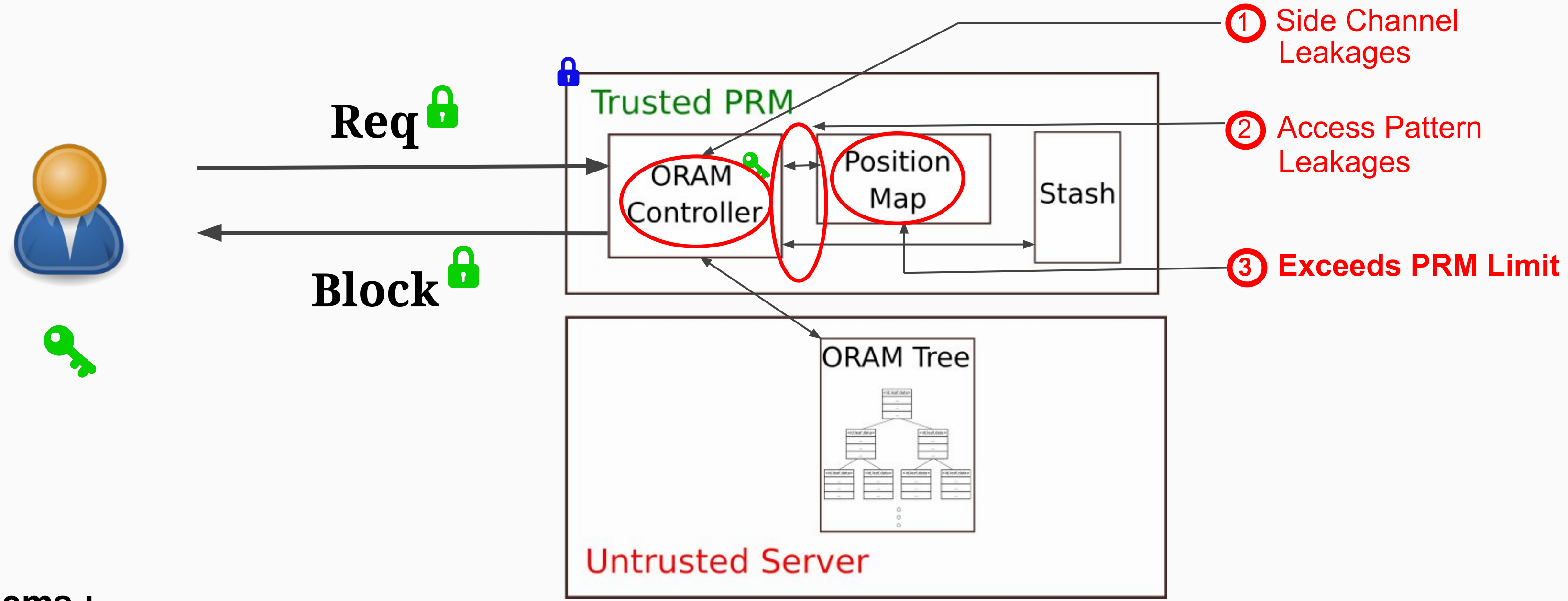
Attempt 1



Problems :

1. Controller code susceptible to side channel leakages
2. **Access pattern leakages of position map and stash accesses**

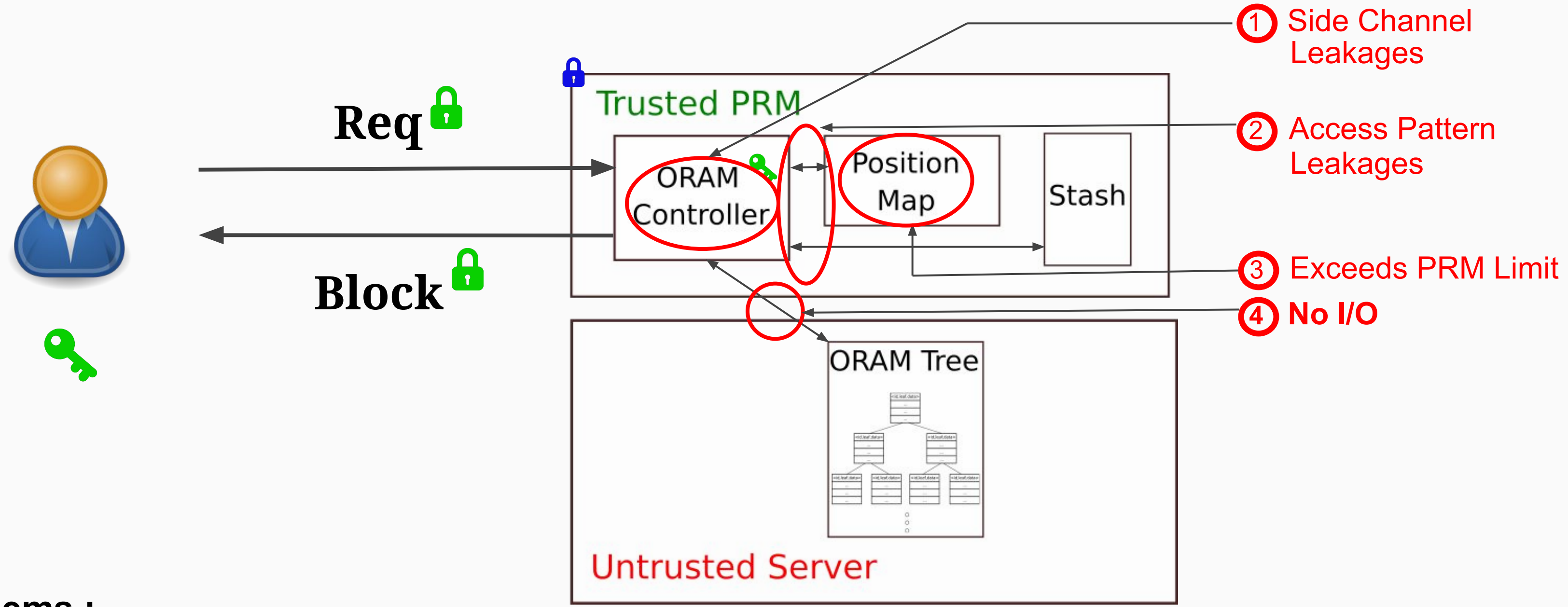
Attempt 1



Problems :

1. Controller code susceptible to side channel leakages
2. Access pattern leakages of position map and stash accesses
3. **Position map could exceed available PRM**

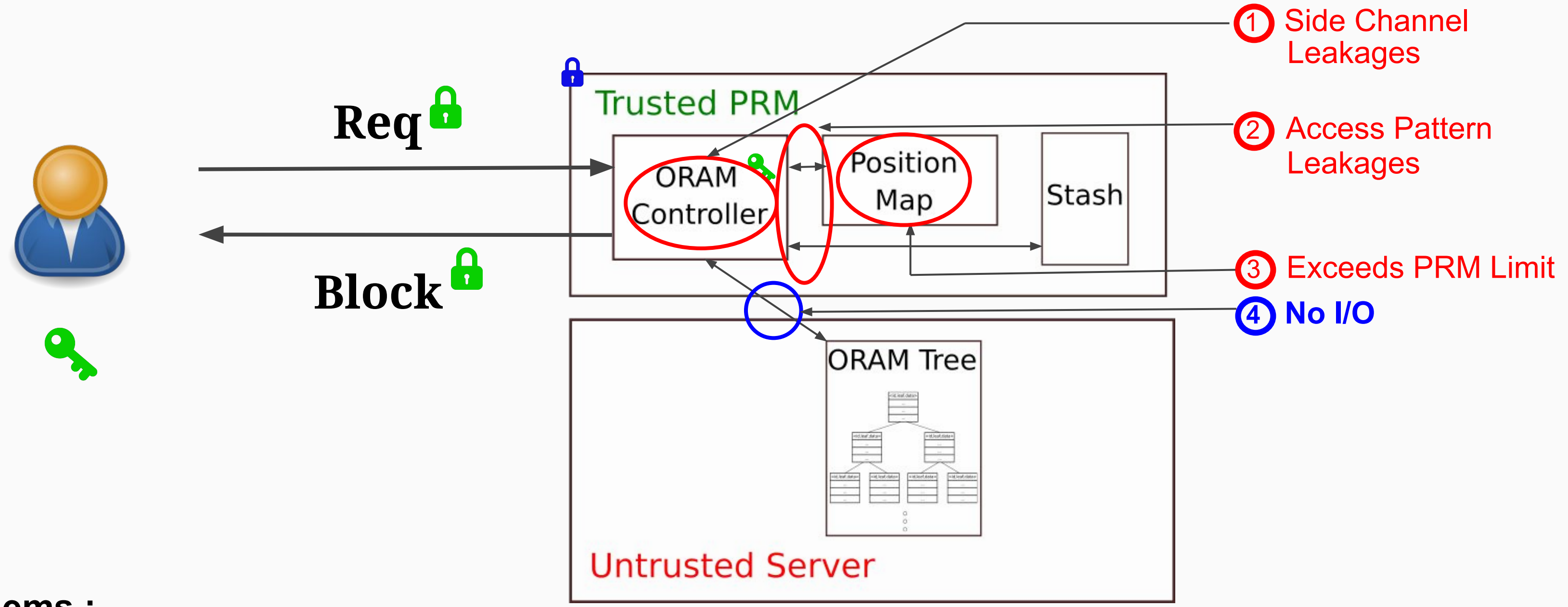
Attempt 1



Problems :

1. Controller code susceptible to side channel leakages
2. Access pattern leakages of position map and stash accesses
3. Position map could exceed available PRM
4. **Enclaves do not have IO support**

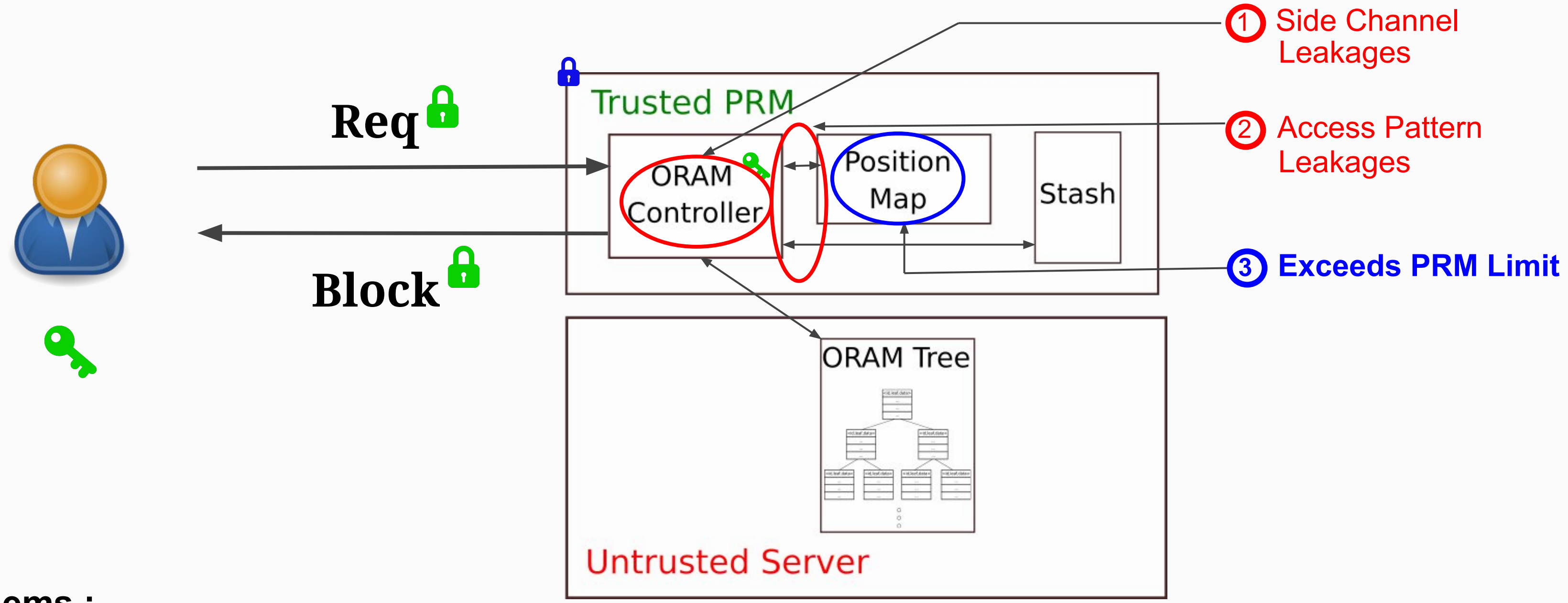
Attempt 2



Problems :

1. Controller code susceptible to side channel leakages
2. Access pattern leakages of position map and stash accesses
3. Position map could exceed available PRM
4. **Enclaves do not have IO support**

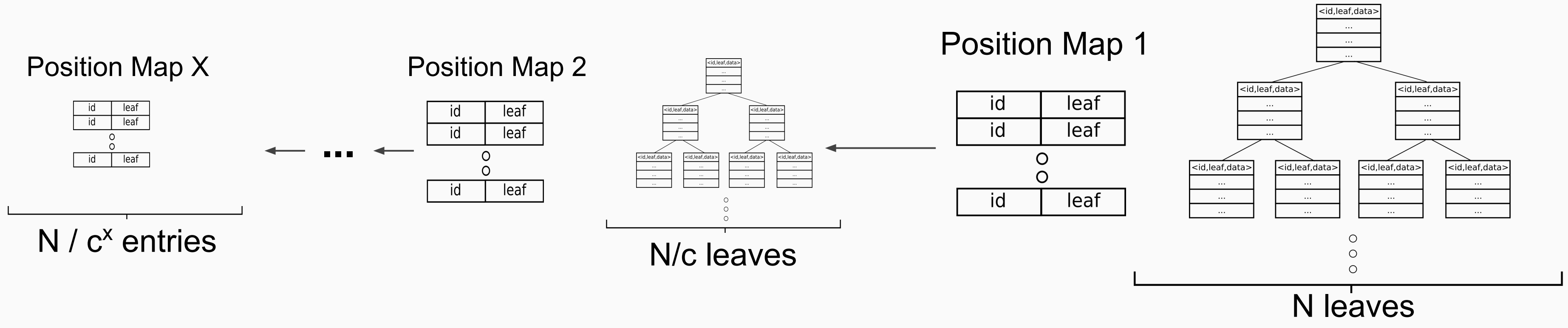
Attempt 2



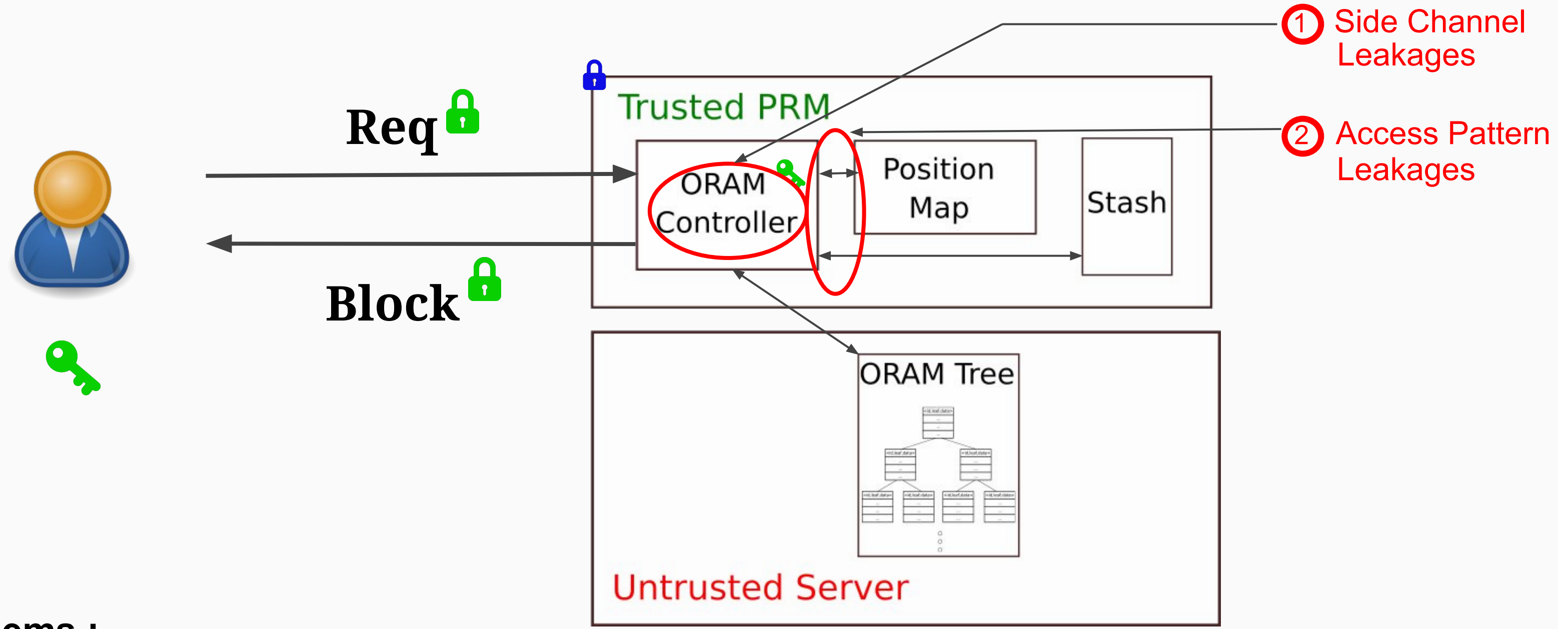
Problems :

1. Controller code susceptible to side channel leakages
2. Access pattern leakages of position map and stash accesses
3. **Position map could exceed available PRM**

Attempt 2 : Recursive ORAMs



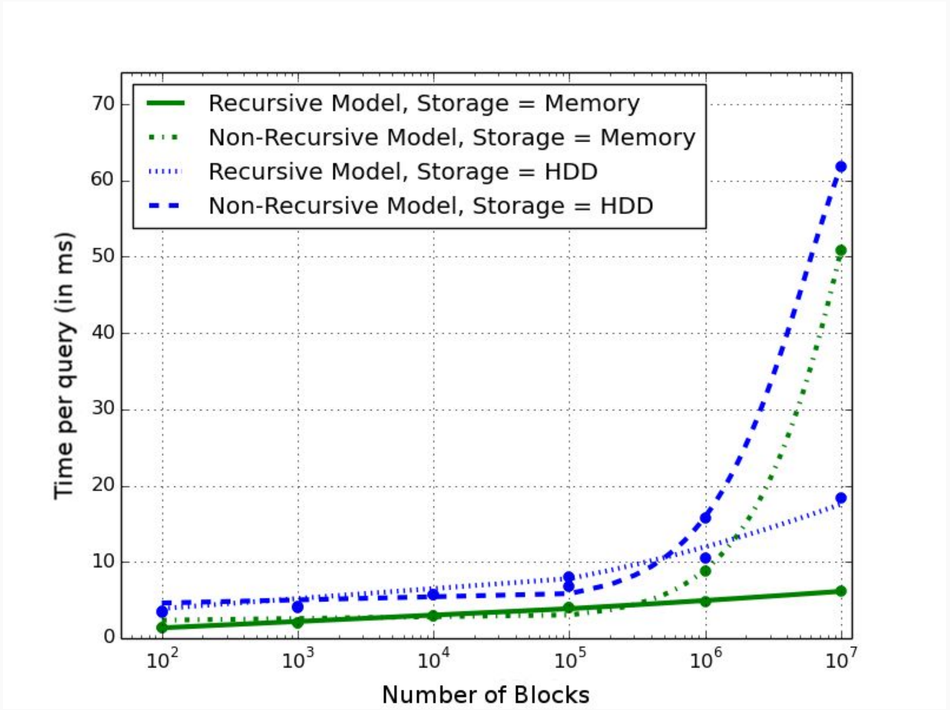
Attempt 2



Problems :

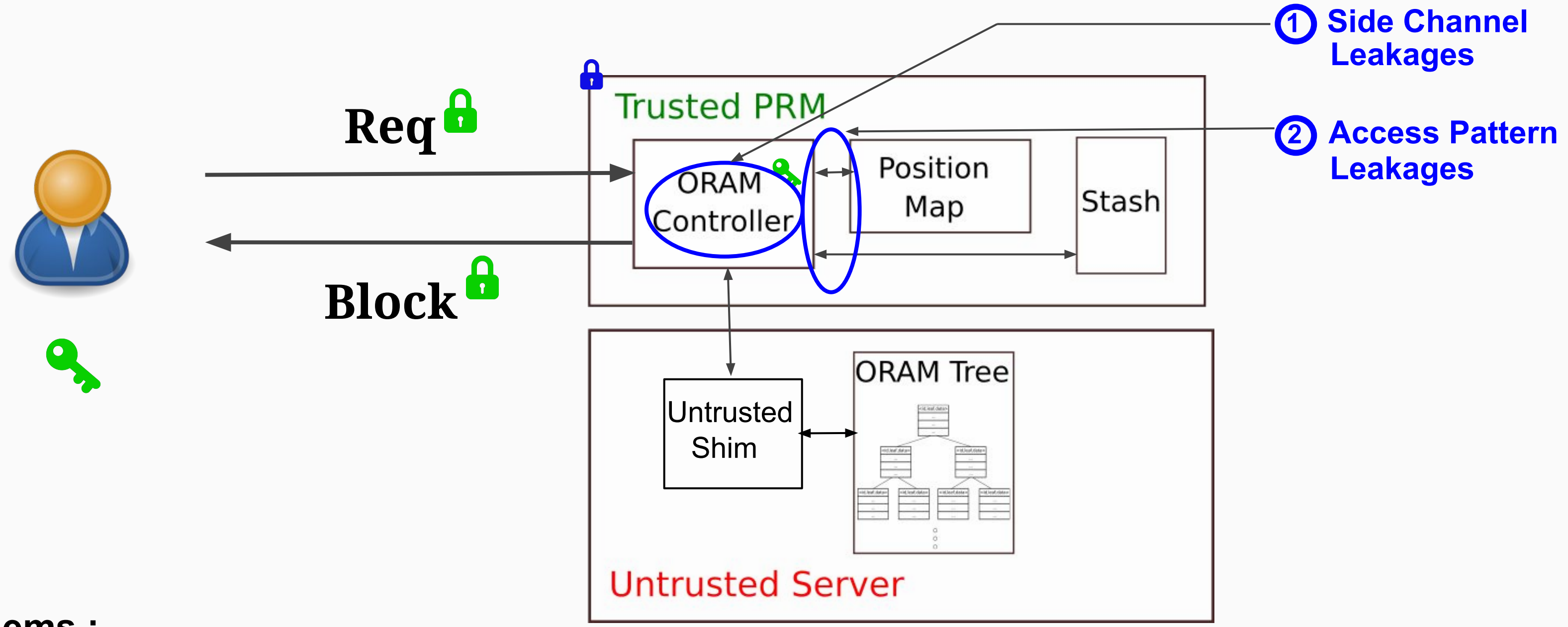
1. Controller code susceptible to side channel leakages
2. Access pattern leakages of position map and stash accesses

Evaluation : Recursion



Data blocks are of 1 KB size in this experiment

Attempt 3 : Solving the security issues



Problems :

1. **Controller code susceptible to side channel leakages**
2. **Access pattern leakages of position map and stash accesses**

Building blocks for side-channel proofing

1) Oblivious functions at assembly level
   - Library of assembly-level functions for oblivious operations.
   - Wrapper functions over CMOV instruction [14,15]
   - Example function :

```
oupdate <srcT, destT> (bool cond, srcT *src, destT *dest, sizeT sz)
```

[14] - Ohrimenko, Olya, et al. "Oblivious Multi-Party Machine Learning on Trusted Processors." *USENIX Security Symposium*. 2016.
[15] - Rane, Ashay, Calvin Lin, and Mohit Tiwari. "Raccoon: Closing Digital Side-Channels through Obfuscated Execution." *USENIX Security Symposium*. 2015.

Building blocks for side-channel proofing

2) Constant time code for the underlying ORAM schema
   - Code branches must be data independent
   - Access to stash and position map are made through linear scans

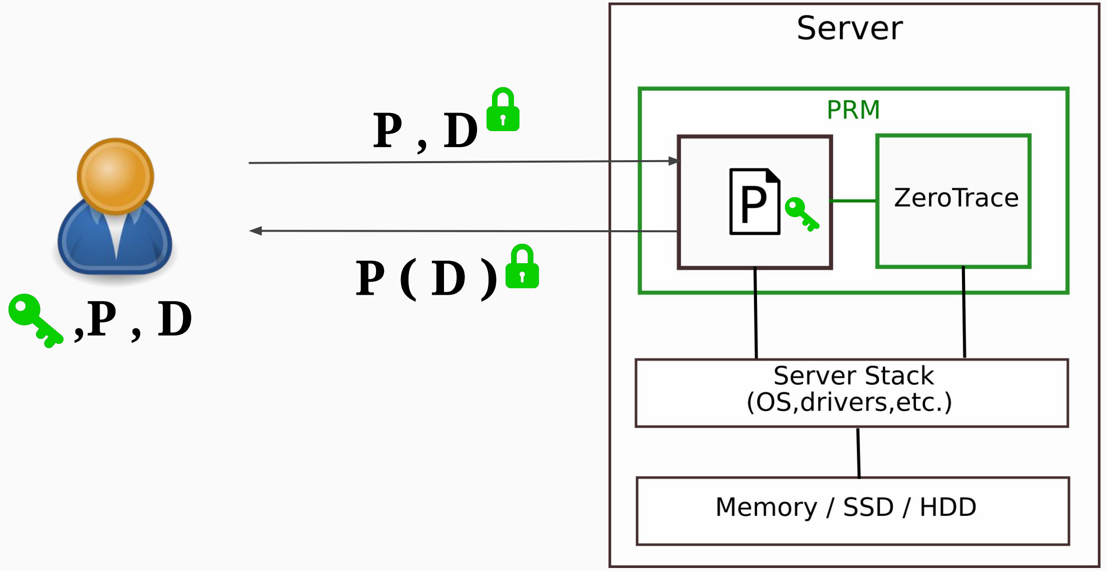
ZeroTrace

ZeroTrace

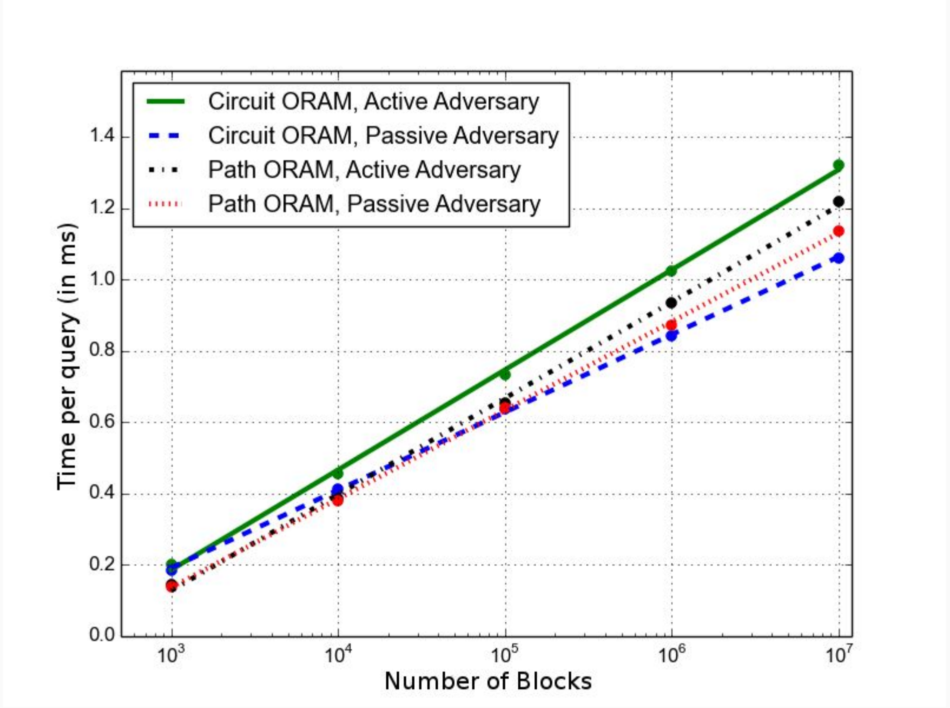
- First oblivious memory controller on a real secure hardware platform
- Flexible storage backends
- ZeroTrace is secure against ALL software side-channel attacks since it realizes the oblivious enclave execution definition.

ZeroTrace - Usage Models

1) Memory Protection for Secure Computation
   - Memory controller for other enclaves
   - Data accesses are now side-channel secure



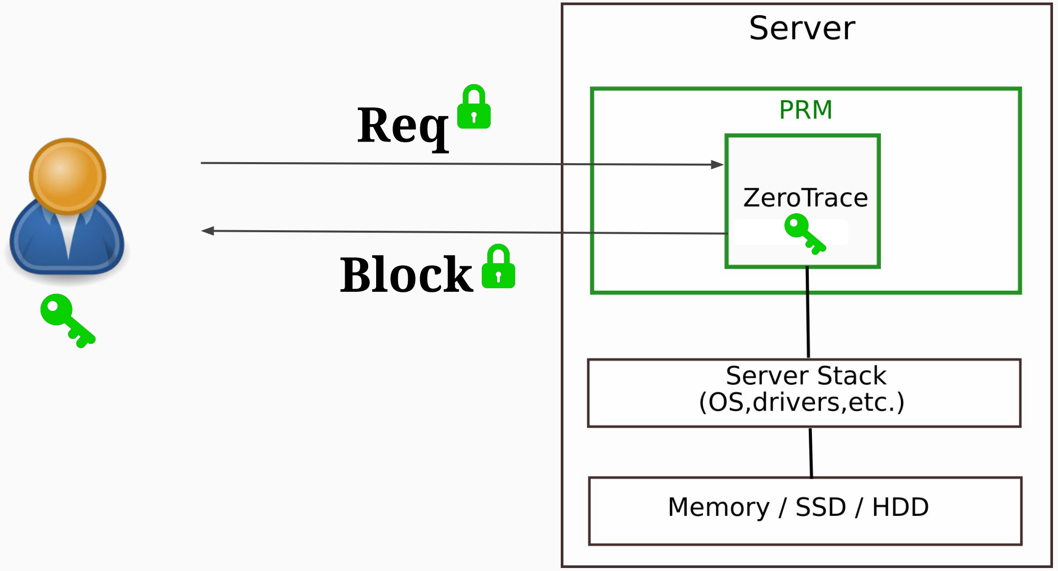
Evaluation : ZeroTrace performance with small data size



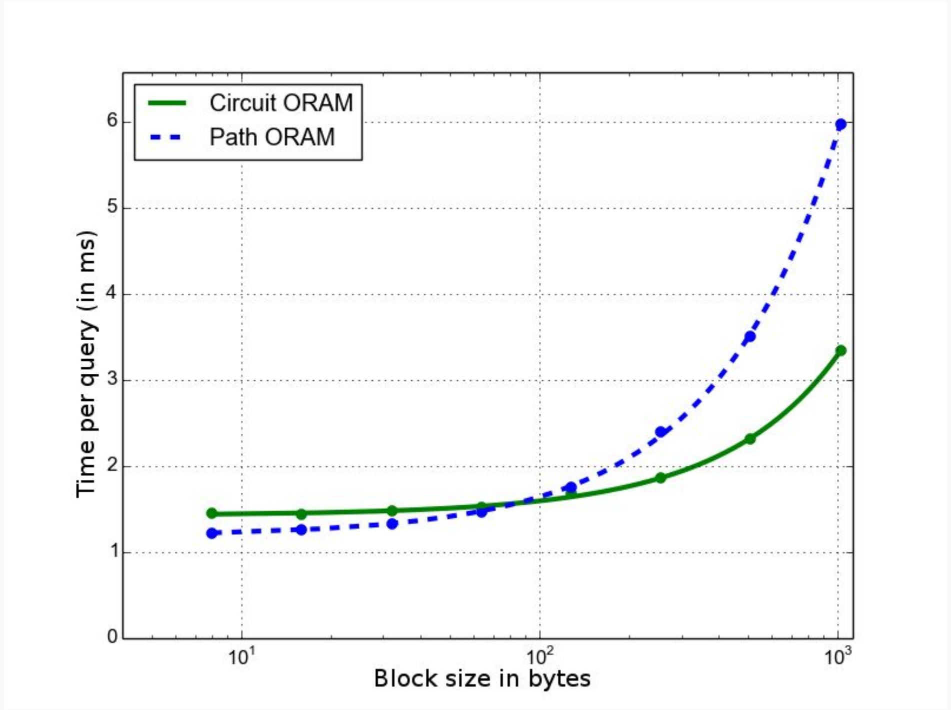
Data blocks are of 8 bytes size in this experiment

ZeroTrace - Usage Models

2) Remote Oblivious Data Storage
   - Order of magnitude network bandwidth saving
   - Order of magnitude decrease in access latency



Evaluation : ZeroTrace performance with increasing data sizes



How to use ZeroTrace

In order to use oblivious memory via ZeroTrace, where necessary :

- Create an oblivious memory abstraction by :
  ZeroTrace_New (label, N, block_size, <params>)

- Access this oblivious memory by :
  ZeroTrace_Access (label, id, op, data*)

Summary

- Illustrated design and evaluation of ZeroTrace
- Showed how to achieve efficient secure remote computation through ZeroTrace
- Go play with ZeroTrace : https://github.com/Sajin7/ZT

Summary

- Illustrated design and evaluation of ZeroTrace
- Showed how to achieve efficient secure remote computation through ZeroTrace
- Go play with ZeroTrace : https://github.com/Sajin7/ZT

Bonus Slides !

Comparing with Hardware ORAM solutions :

- No deployed / practically available solution
  Since H/W required is custom and not commercially available
- Typically tied to DRAM storage
- All or nothing , no flexibility

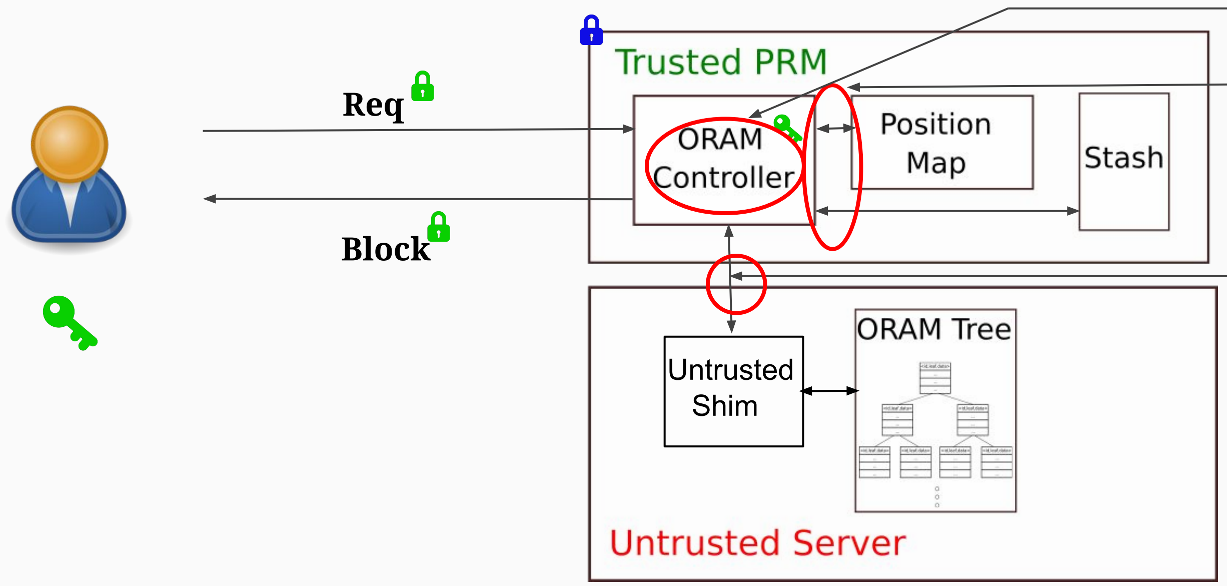
Comparing with Raccoon:

- Experiments on Xeon processors (No SGX Support)
- Parameterized to fit recursion within the register space, and discarded recursive ORAMs for SGX setting
- Streaming doesn’t account for encryption/decryption overhead

How does Meltdown/Spectre affect ZeroTrace

- Meltdown does not effect ZeroTrace. No PoC currently
- Spectre_1 doesn’t pan out since there are no branches
- Spectre_2 has been patched by Intel already
- We are still investigating this

High-level Security of ZeroTrace



Side-channel proofed using oblivious techniques

Inherits traditional ORAM security

Future Steps:

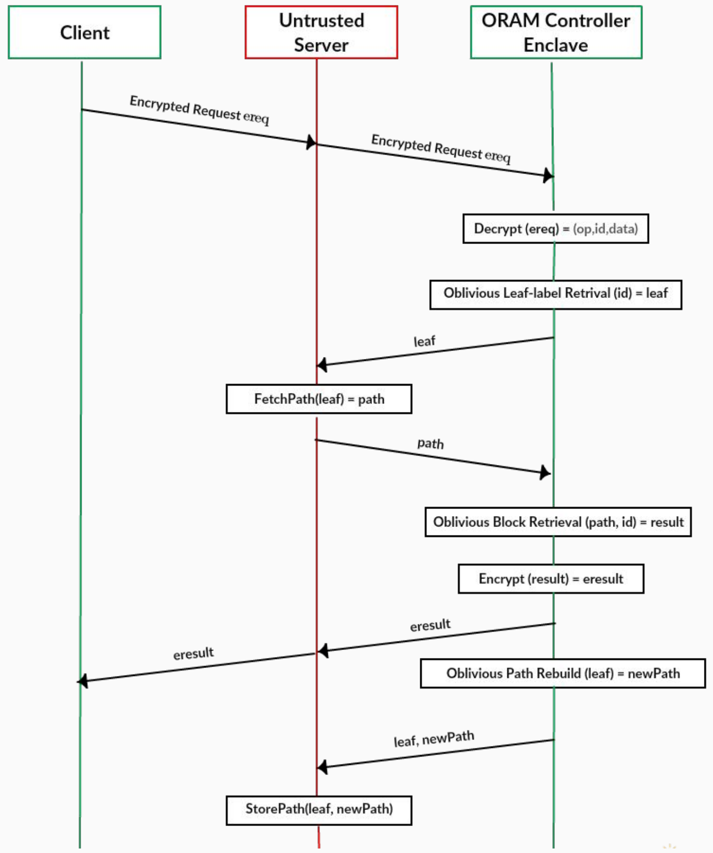
- Deploying ZeroTrace as an open source library
- Optimizing data structure support
- Optimizing initialization costs
- Asynchronous ORAM

Desired Property : Oblivious Enclave Execution

When a program P is loaded in an enclave, and a set of inputs $\mathbf{y} = (\mathrm{in}_1,\ldots,\mathrm{in}_M)$ are executed by this enclave it results in an adversarial view **V**($\mathbf{y}$) = **trace**($(\mathbf{E_p},\mathrm{in}_1),\ldots,(\mathbf{E_p},\mathrm{in}_M)$). We say that the enclave execution is oblivious if given two sets of inputs $\mathbf{y}$ and $\mathbf{z}$, their adversarial views **V**($\mathbf{y}$) and **V**($\mathbf{z}$) are computationally indistinguishable.

Here **trace**($\mathbf{E_p},\mathrm{in}$) captures the execution trace induced by running the enclave $\mathbf{E_p}$ with input in. This **trace**($\mathbf{E_p},\mathrm{in}$) contains all the powerful side channel artifacts that the adversary can view such as cache usage, page faults, etc.

ZeroTrace - Security Arguement


Client
Untrusted Server
ORAM Controller Enclave
Encrypted Request ereq
Encrypted Request ereq
Decrypt (ereq) = (op,id,data)
Oblivious Leaf-label Retrival (id) = leaf
leaf
FetchPath(leaf) = path
path
Oblivious Block Retrieval (path, id) = result
Encrypt (result) = eresult
eresult
eresult
Oblivious Path Rebuild (leaf) = newPath
leaf, newPath
StorePath(leaf, newPath)

Side-channel proofed leaf label retrieval

Non-Oblivious Leaf-label Retrieval :

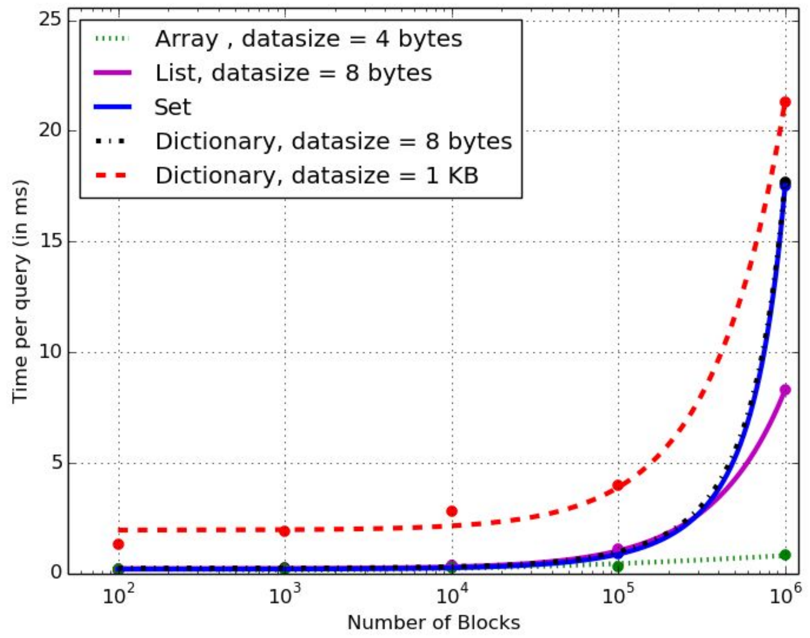
```
newleaf = random(N)
leaf = position_map[x]
position_map[x] = newleaf
```

Oblivious Leaf-label Retrieval :

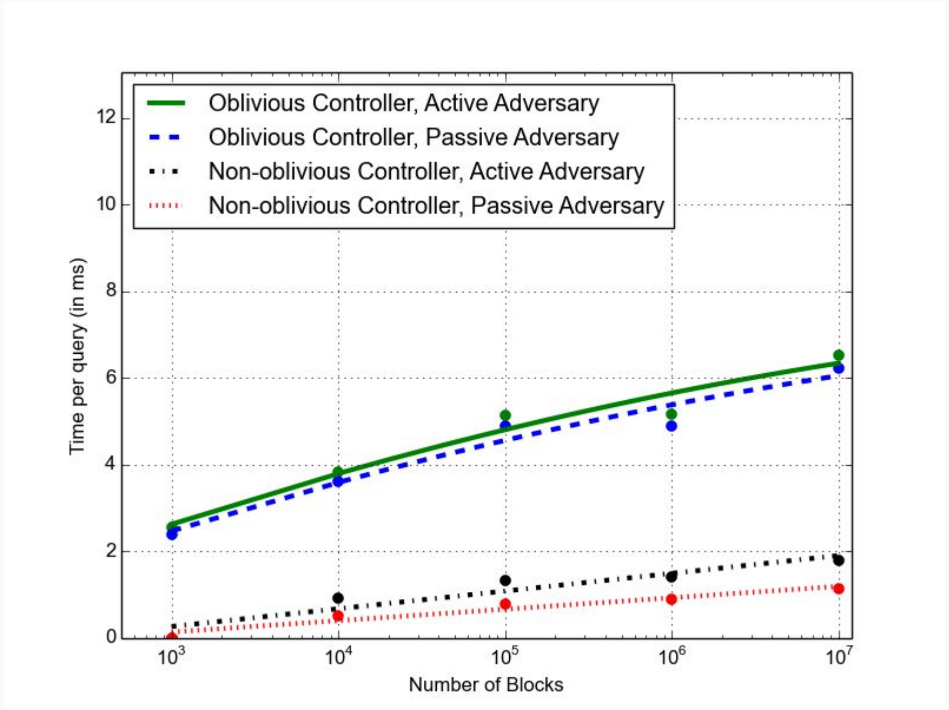
```
newleaf = random(N)
for i in range(0, N):
  cond = (i == x)
  oupdate(cond, position_map[i], leaf, size)
  oupdate(cond, newleaf, position_map[i], size)
```

Other Graphs

Evaluation : Memory Primitives

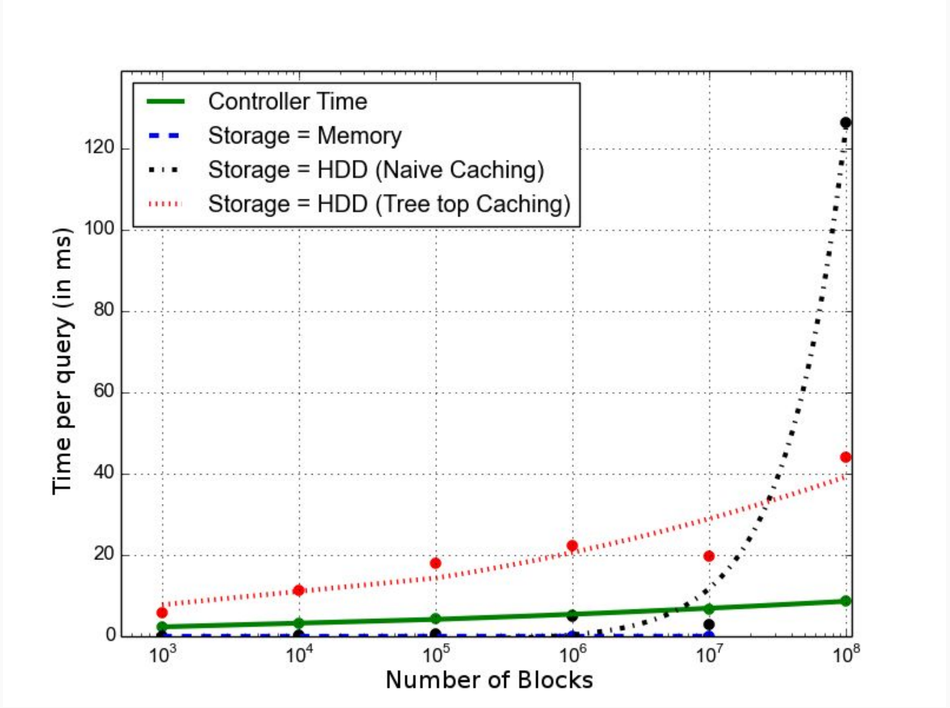


Evaluation : Flexibility of Controller

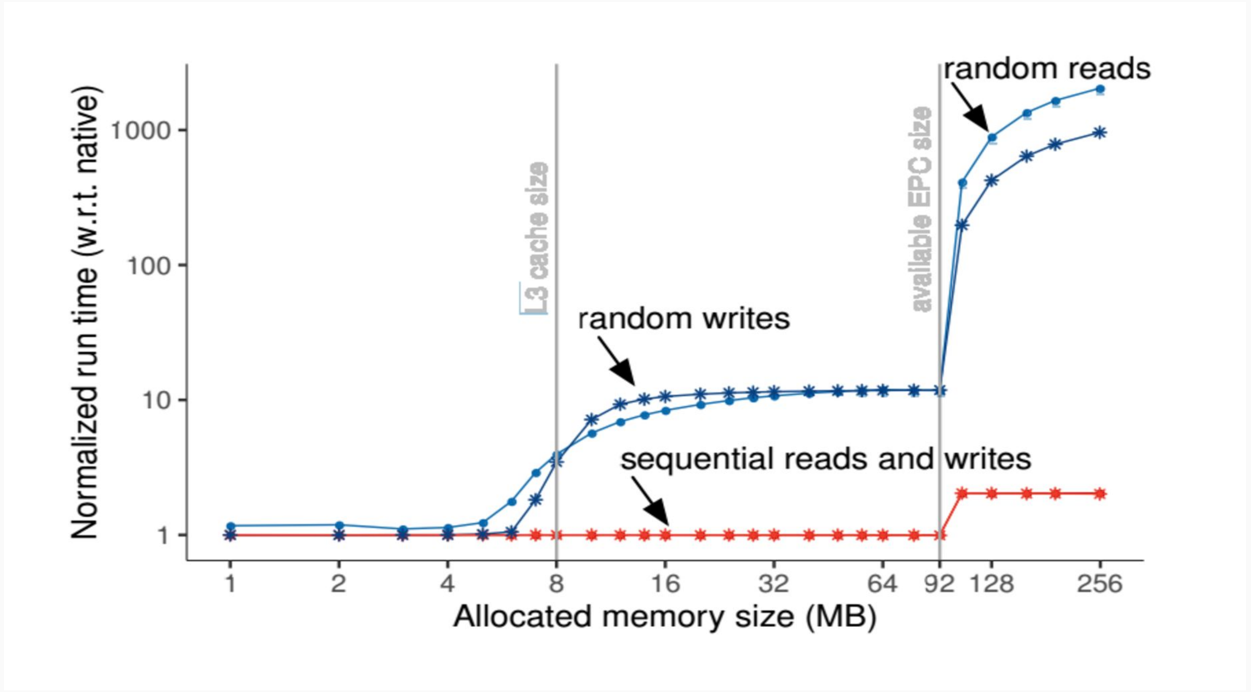


Data blocks are of 1 KB size in this experiment

Evaluation : Breakdown of Request Time



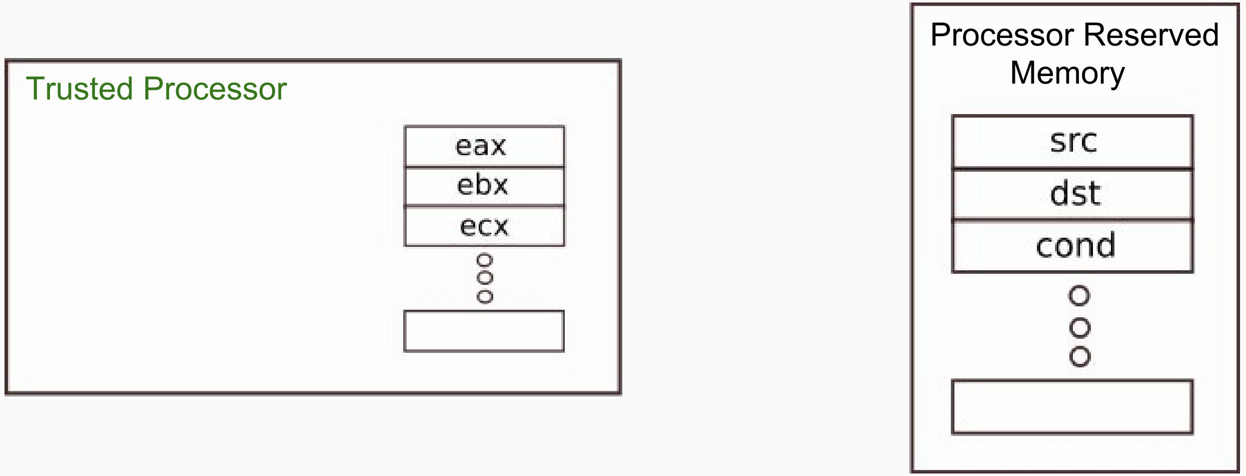
Overhead of EPC memory accesses



oupdate() in depth

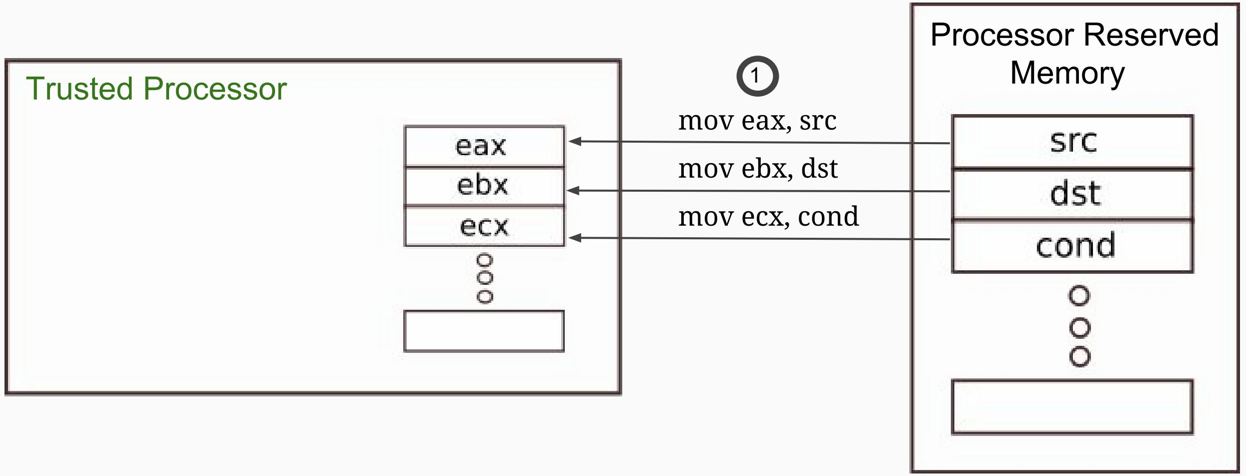
Building blocks for side-channel proofing

```
oupdate <srcT, destT> (bool cond, uint32_t src, uint32_t dest, sizeT sz)
```



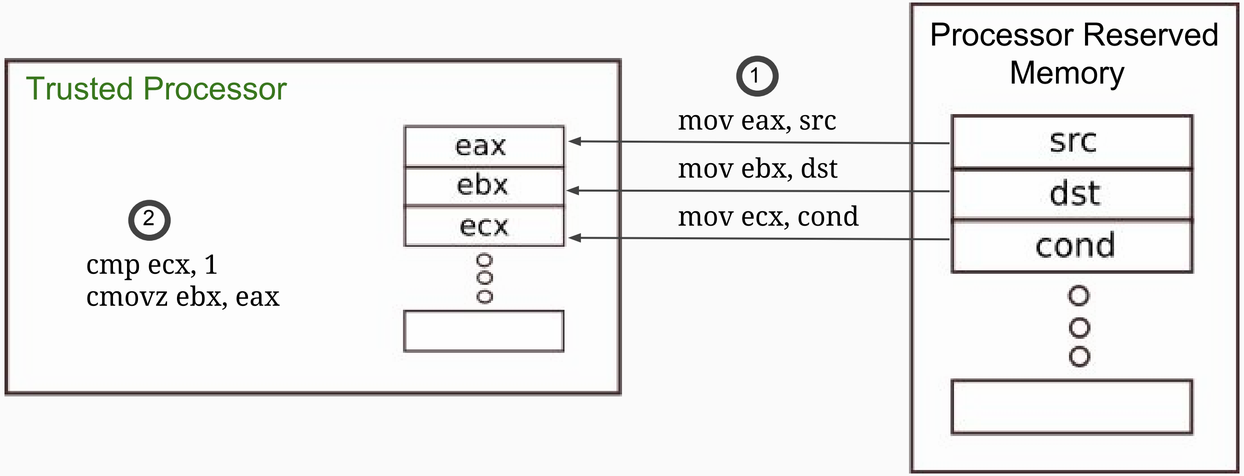
Building blocks for side-channel proofing

```
oupdate <srcT, destT> (bool cond, uint32_t src, uint32_t dest, sizeT sz)
```



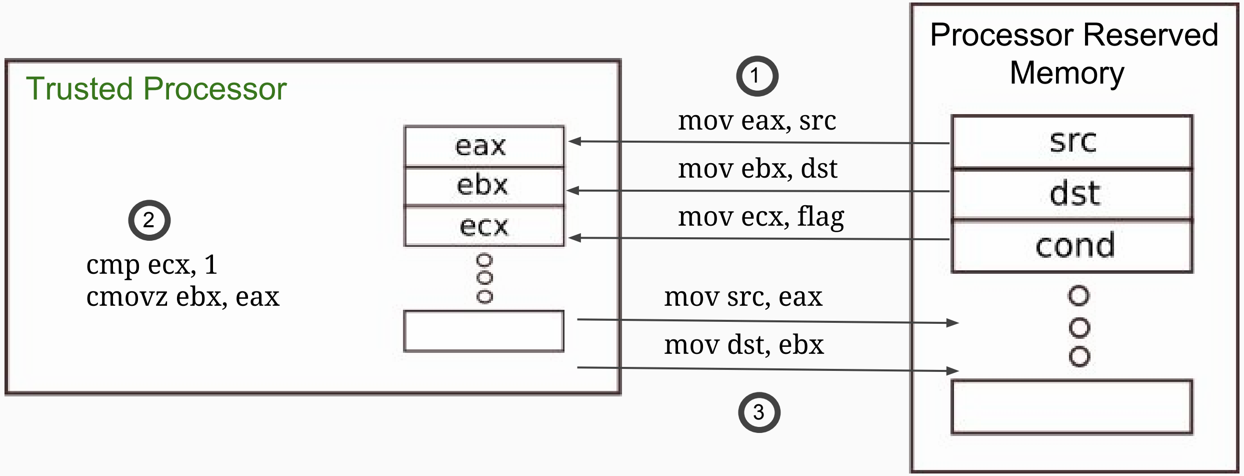
Building blocks for side-channel proofing

```
oupdate <srcT, destT> (bool cond, uint32_t src, uint32_t dest, sizeT sz)
```



Building blocks for side-channel proofing

```
oupdate <srcT, destT> (bool cond, uint32_t src, uint32_t dest, sizeT sz)
```



Access Protocol for Tree based ORAM schemes

Tree based ORAMs - Access

① Fetch leaf for id = 7 from Position Map

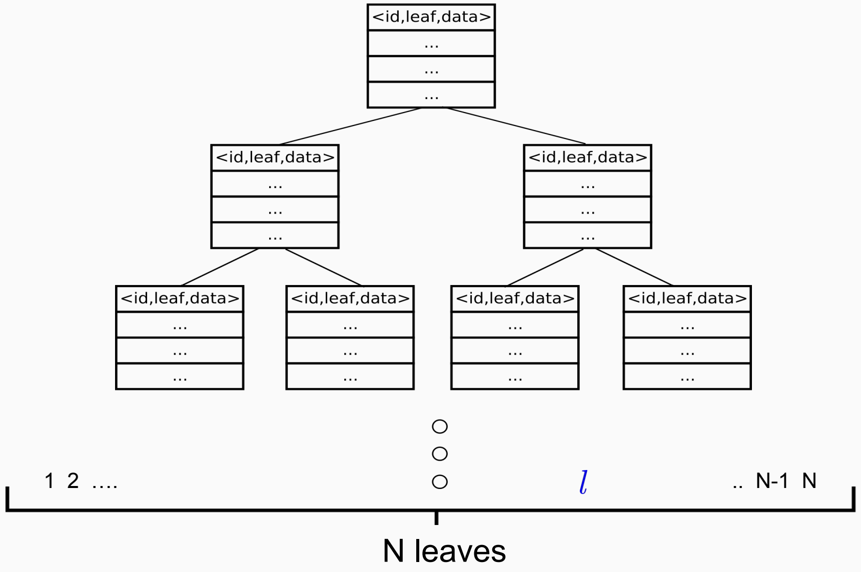
Client

Request ID : 7

Position Map

id	leaf
7	l
○	
○	
id	leaf

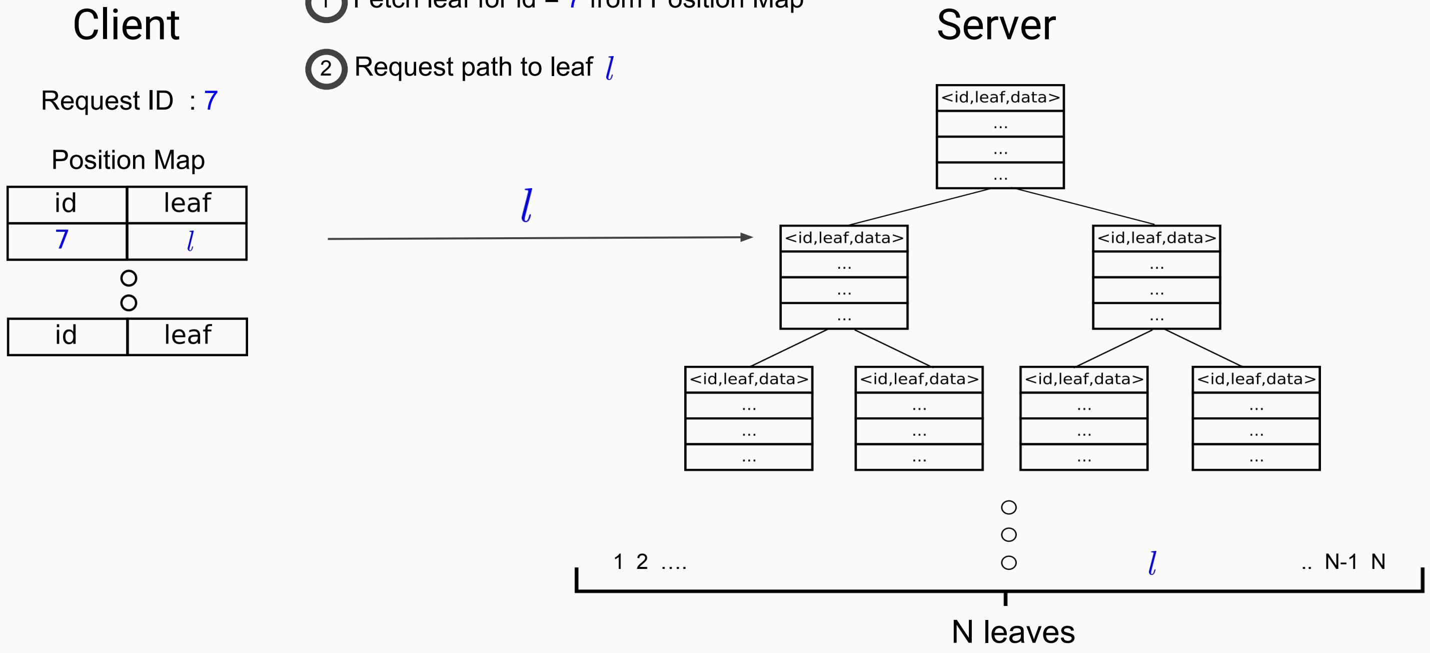
Server



Tree based ORAMs - Access

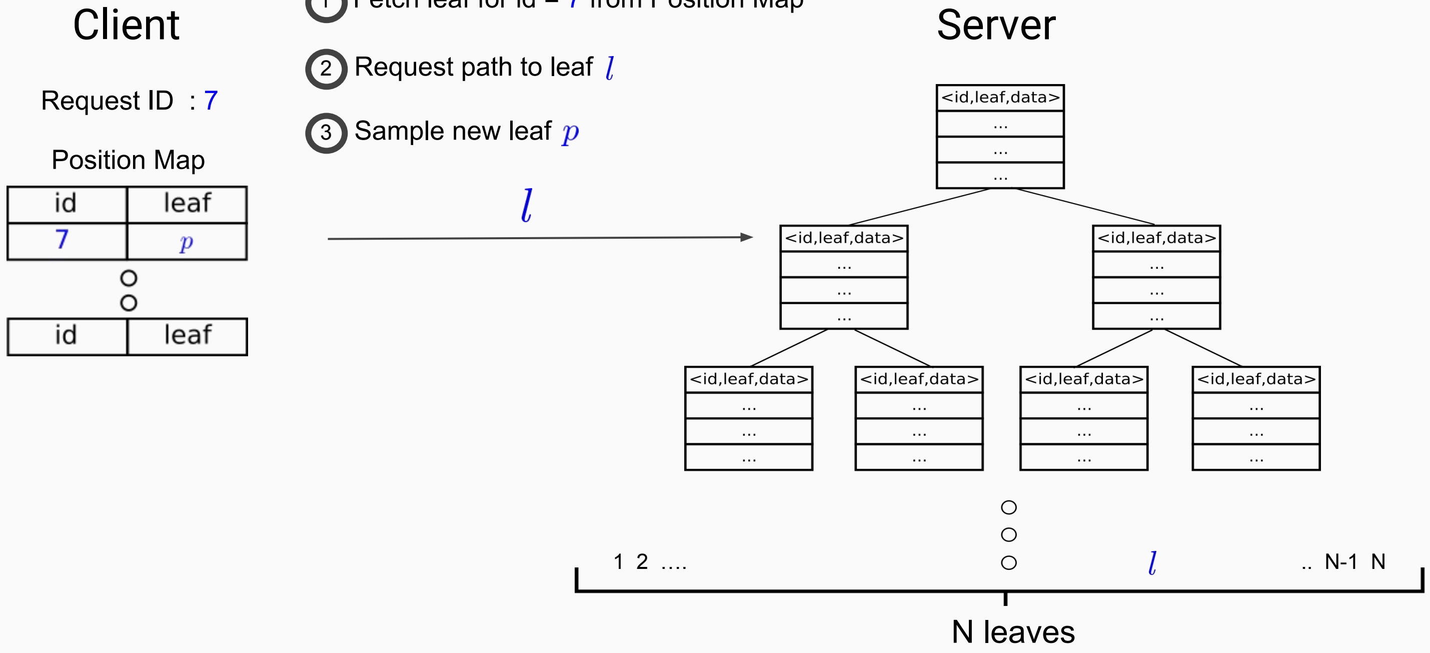
① Fetch leaf for id = 7 from Position Map

② Request path to leaf l

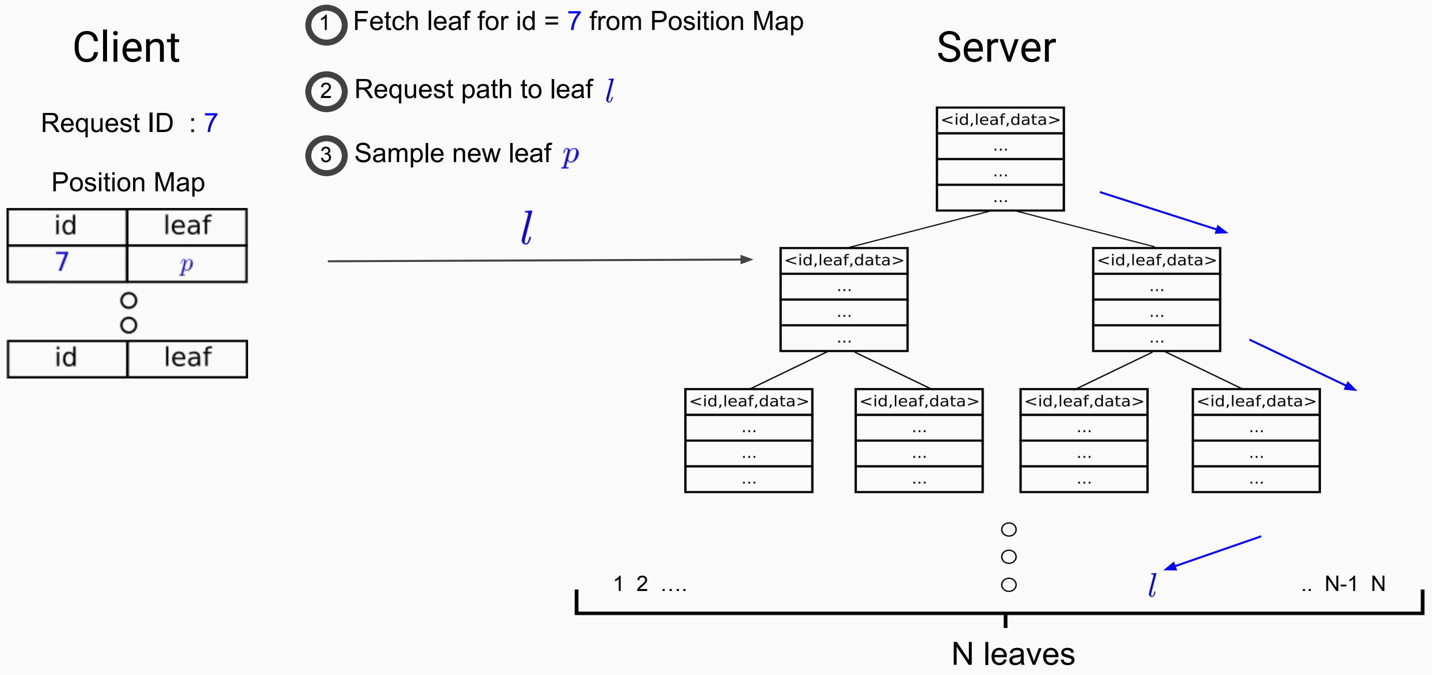


Tree based ORAMs - Access

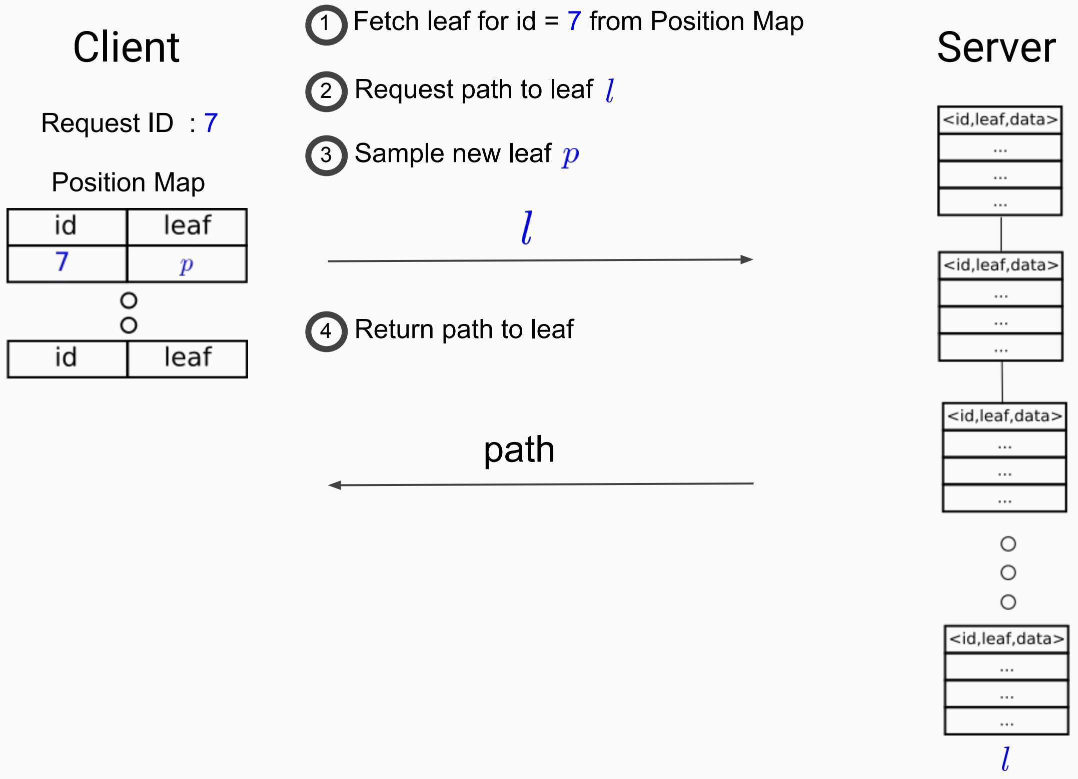
1. Fetch leaf for id = 7 from Position Map
2. Request path to leaf l
3. Sample new leaf p



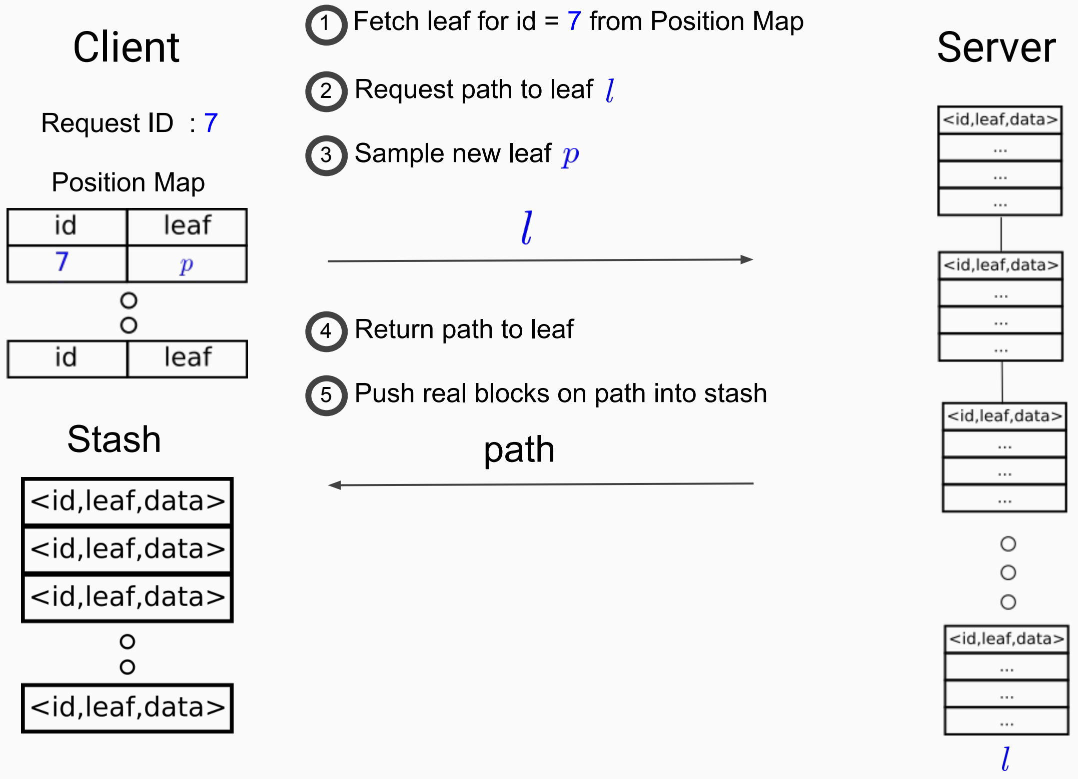
Tree based ORAMs - Access



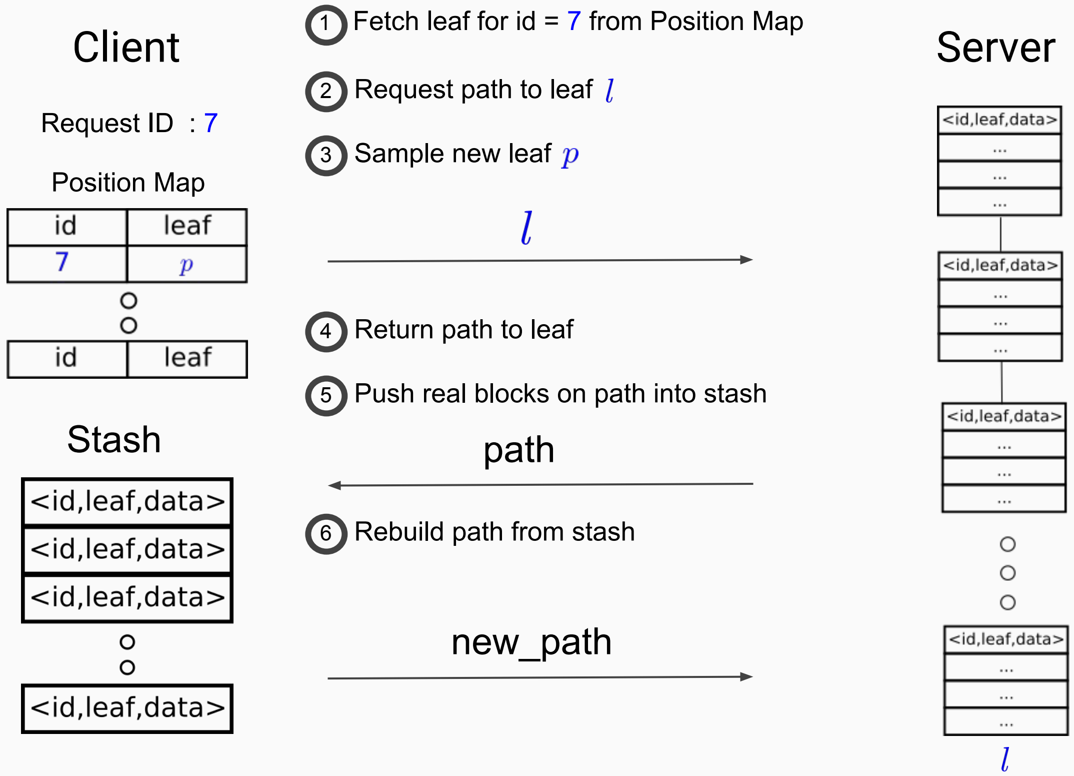
Tree based ORAMs - Access



Tree based ORAMs - Access

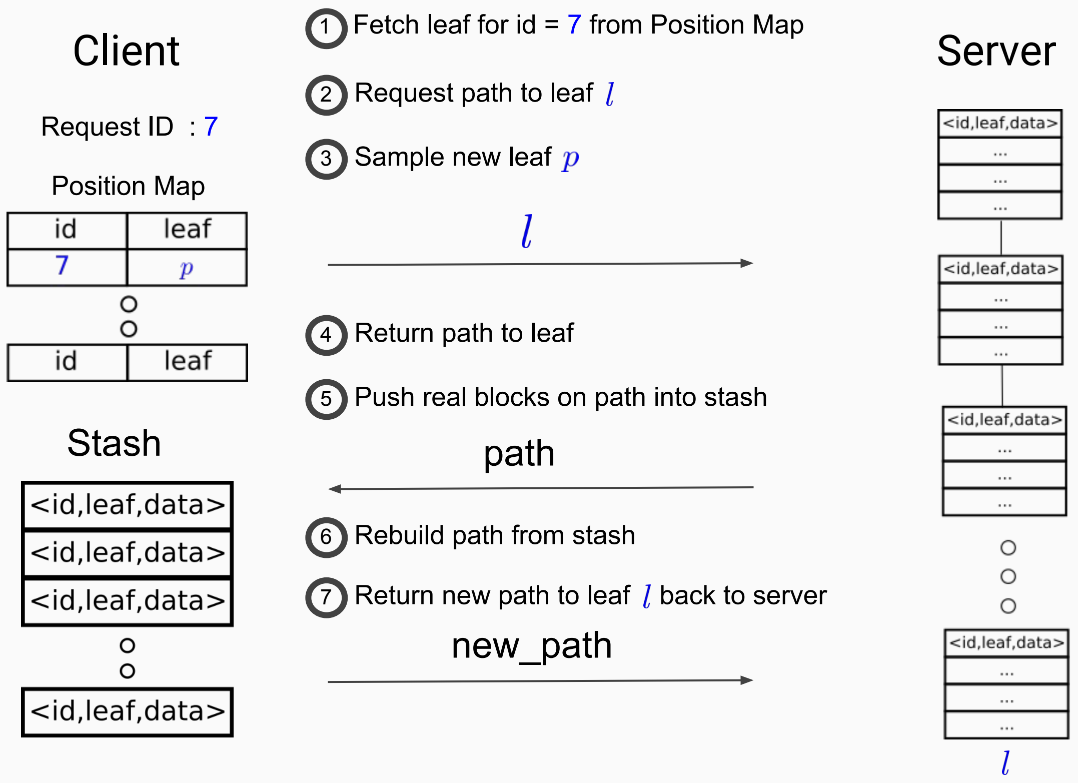


Tree based ORAMs - Access



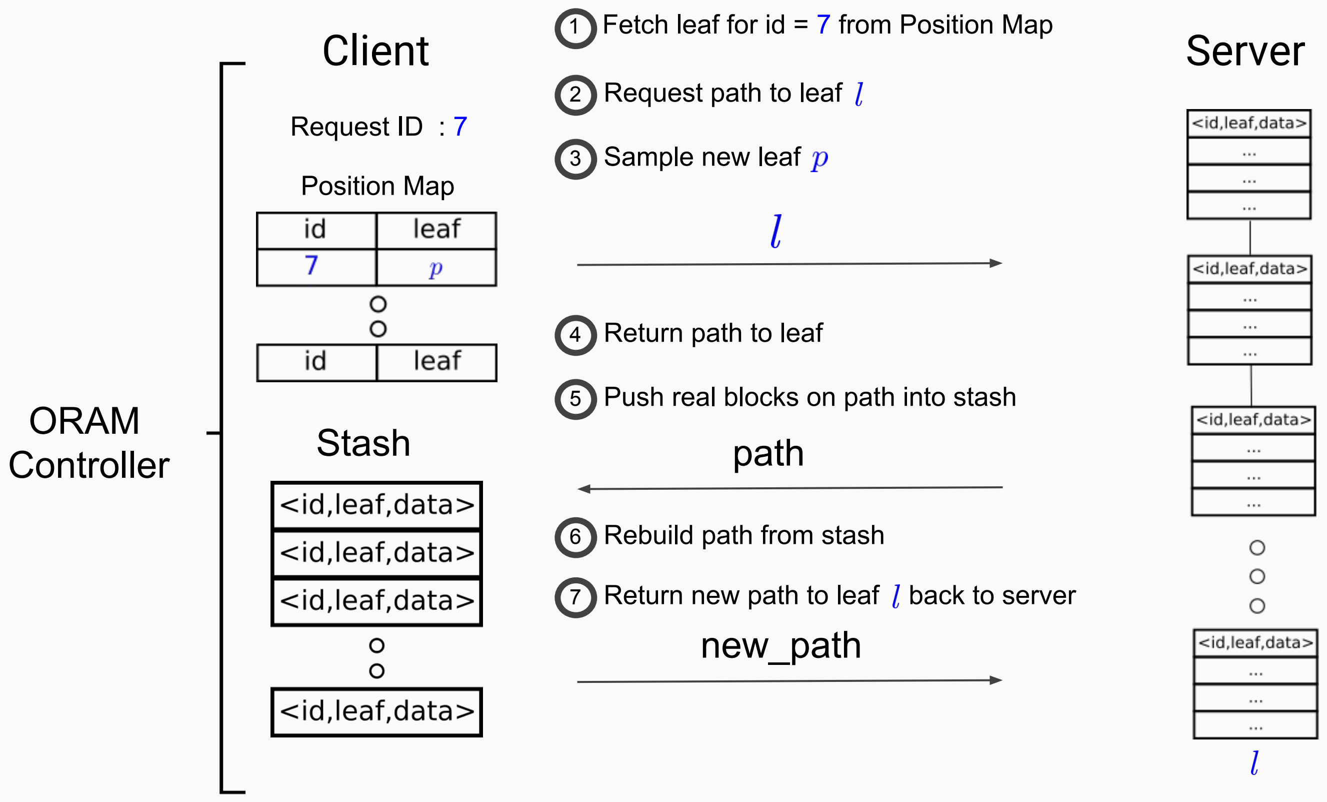
1. Fetch leaf for id = 7 from Position Map
2. Request path to leaf l
3. Sample new leaf p
4. Return path to leaf
5. Push real blocks on path into stash
6. Rebuild path from stash

Tree based ORAMs - Access



1. Fetch leaf for id = 7 from Position Map
2. Request path to leaf l
3. Sample new leaf p
4. Return path to leaf
5. Push real blocks on path into stash
6. Rebuild path from stash
7. Return new path to leaf l back to server

Tree based ORAMs - Access



1. Fetch leaf for id = 7 from Position Map
2. Request path to leaf l
3. Sample new leaf p
4. Return path to leaf
5. Push real blocks on path into stash
6. Rebuild path from stash
7. Return new path to leaf l back to server